



\_\_\_\_\_1\_\_\_\_\_ \_P\_. \_@| | | | \_\_\_\_\_\$S]\_\$\_`\_ \$T&\_\_\_\_\_K\_\_\_\_\_H\_\_\_\_\_

REV7\_SPEC\_\_\_\_\_

\_\_\_\_\_

1

EAGLE  
Architecture

Author: Steven J. Wallach x6962 (Westboro)  
Revision: 7  
Document: 009000180-7  
Attachments: Scientific Instruction Set as Appendix III

10:18:3

Data General Corporation  
Company Confidential

Table of Contents

Preface . . . . .	p-1
Chapter 1--Overview . . . . .	1-1
1.1 Introduction . . . . .	1-1
1.2 EAGLE Architecture Summary . . . . .	1-1
1.3 EAGLE/ECLIPSE Relationship . . . . .	1-3
Chapter 2--Data Types and Formats . . . . .	2-1
2.1 Basic Allowable Types . . . . .	2-1
2.1.1 Floating Point . . . . .	2-1
2.1.2 Fixed . . . . .	2-2
2.1.3 Boolean . . . . .	2-2
2.1.4 String . . . . .	2-3
2.1.5 Commercial . . . . .	2-3
2.2 Floating Point Rounding . . . . .	2-5
Chapter 3--REGISTER SET AND RESERVED MEMORY LOCATIONS . . . . .	3-1
3.1 Register Set . . . . .	3-1
3.1.1 Fixed Point . . . . .	3-1
3.1.2 Floating Point Registers . . . . .	3-3
3.1.3 Stack Management . . . . .	3-5
3.1.4 Memor_	
y Management . . . . .	3-6
3.2 Page 0 Memory Locations . . . . .	3-7
3.2.1 Segment 0 - Page 0 . . . . .	3-8
3.2.2 Segment 1,2,3,4,5,6,7 - Page 0 . . . . .	3-13
Chapter 4--Logical Address Space . . . . .	4-1
4.1 Generation of a Word Logical Address . . . . .	4-2
4.2 Generation of a Byte Logical Address . . . . .	4-3
4.3 Generation of a Bit Logical Address . . . . .	4-4
Chapter 5--Memory Management . . . . .	5-1

11:34:52  
17/Sep/80

Data General Corporation  
Company Confidential

5.1	Introduction . . . . .	5-1
5.2	Logical to Physical Translation . . . . .	5-1
5.3	Page Table Entry . . . . .	5-3
5.4	Physical Page Management . . . . .	5-10
5.5	Physical Addresses/ Power Up - I/O RESET . . . . .	5-10
5.6	Address Translation Unit (ATU) . . . . .	5-11
5.6.1	Process Multiplexing . . . . .	5-12
Chapter 6--Protection System . . . . .		6-1
6.1	Protection Notes . . . . .	6-4
6.2	Indirection . . . . .	6-5
6.3	SBR Validity Bit Protection . . . . .	6-6
6.4	Indirect Chain Protection . . . . .	6-6
6.5	LEF Protection . . . . .	6-7
6.6	I/O Protection . . . . .	6-7
6.7	Gate Access/Ring Crossing . . . . .	6-7
6.7_		
6.7.1	Initiation and Mediation of Ring Crossing . . . . .	6-7
6.7.2	Stack Switching and Argument Pushing . . . . .	6-10
6.7.3	Trojan Horse Pointers . . . . .	6-11
Chapter 7--THE EAGLE INSTRUCTION SET . . . . .		7-1
7.1	Fixed Point Overflow . . . . .	7-3
7.2	Fixed Point Indexed Address Instructions . . . . .	7-4
7.3	Stack Control . . . . .	7-11
7.4	Fixed Point Word Arithmetics . . . . .	7-14
7.5	Fixed Point Double Word Arithmetics . . . . .	7-16
7.6	Fixed Point Double Word Logicals . . . . .	7-23
7.7	Double Word Compare ACs . . . . .	7-26
7.8	Fixed Point Word Indexed Address Arithmetics . . . . .	7-27
7.9	Fixed Point Double Word Indexed Address Arithmetics . . . . .	7-30
7.10	Skip Immediates . . . . .	7-33
7.11	Memory Add/Sub Short Immediate . . . . .	7-34
7.12	DO LOOP . . . . .	7-37
7.13	Program Flow . . . . .	7-39
7.14	Double Word Bit Operations . . . . .	7-43
7.15	Character Instructions . . . . .	7-44
7.16	Skip on Word Masked Field . . . . .	7-45
7.17	Skip on Double Word Masked Field . . . . .	7-46
7.18	Processor Status Register and Carry Instructions . . . . .	7-49
7.19	Others . . . . .	7-51
7.20	Argument Validation/Physical Address . . . . .	7-60
Chapter 8--Subroutine Call/Save/Return - Interrupt Return . . . . .		8-1

Data General Corporation  
Company Confidential

Chapter 9--QUEUE INSTRUCTIONS . . . . .	9-1
Chapter 10--FLOATING POINT INSTRUCTIONS . . . . .	10-1
10.1 Floating Point Arithmetic Algorithms . . . . .	10-1
10.1.1 Floating Point Rounding . . . . .	10-1
10.1.1.1 Truncation . . . . .	10-2
10.1.1.2 Unbiased Round . . . . .	10-2
10.2 Floating Point Indexed Address Instructions . . . . .	10-3
10.3 Floating Point Status Register . . . . .	10-5
10.4 Conversions: Float/Fix, Integerize, Double/Single . . . . .	10-8
10.5 C/350 Instructions . . . . .	10-10
Chapter 11--DECIMAL NUMERICS . . . . .	11-1
11.1 Instruction Set . . . . .	11-1
11.2 Legal Commercial Data Types 0 to 5 . . . . .	11-3
Chapter 12--PRIVILEGED INSTRUCTIONS . . . . .	12-1
Chapter 13--EAGLE I/O INSTRUCTIONS . . . . .	13-1
13.1 I/O Maps and WLMP Instruction . . . . .	13-3
13.2 IOSB Commands and Instructions . . . . .	13-6
Chapter 14--Interrupts . . . . .	14-1
14.1 Interrupt Mechanism . . . . .	14-1
14.1.1 Immediate Interrupt . . . . .	14-2
14.1.2 Vectored Interrupt . . . . .	14-2
14.1.2.1 Base Level . . . . .	14-3
14.1.2.1.1 Ring 0-Base Level . . . . .	14-3
14.1.2.1.2 Non-Ring 0 -Base Level . . . . .	14-4
14.1.2.2 Intermediate Level . . . . .	14-4
14.1.2.2.1 Intermediate Level - Ring 0 . . . . .	14-4
14.1.2.2.2 Intermediate Level - Non Ring 0 . . . . .	14-5
14.1.2.3 Common Interr_	
upt Epilogue . . . . .	14-5
14.2 Micro Interrupts . . . . .	14-8

11:34:52  
17/Sep/80

Data General Corporation  
Company Confidential



Chapter 15--Fault Mechanism . . . . .	15-1
15.1 Stack Fault . . . . .	15-2
15.2 Protection Fault . . . . .	15-4
15.3 Floating Point Faults . . . . .	15-5
15.3.1 Asynchronous Floating Point Units . . . . .	15-6
15.4 Commercial Faults . . . . .	15-7
15.4.1 WLSN/LDIX/STIX/WLDIX/WSTIX . . . . .	15-7
15.4.2 EDIT and WEDIT . . . . .	15-8
15.5 Page Fault . . . . .	15-11
15.6 Fixed Point Overflow . . . . .	15-12
 Chapter 16--Unimplemented Instructions . . . . .	 16-1
16.1 Undefined Opcodes . . . . .	16-2
 Chapter 17--I/O Specification . . . . .	 17-1
17.1 Programmed I/O . . . . .	17-1
17.2 Data Channel I/O . . . . .	17-2
17.3 Burst Multiplexor I/O . . . . .	17-3
17.4 System Cache . . . . .	17-3
17.5 I/O Processor . . . . .	17-4
17.6 Multiprocessor Link . . . . .	17-4
 Chapter 18--FAIL SAFE TECHNIQUES . . . . .	 18-1
 Chapter 19--ANOMALIES . . . . .	 19-1
19.1 No Load - Skip, XOP, and XOP1 Instructions . . . . .	1_
____(____)9-1	
19.2 Page 0 Addresses . . . . .	19-1
19.3 PC Wraparound . . . . .	19-1
19.4 Float/Fixed Conversions . . . . .	19-2
19.5 Address Wraparound . . . . .	19-2
19.6 Eclipse Signed Divide . . . . .	19-2
19.7 Eclipse Vector & NIO . . . . .	19-2
19.8 Floating Point Fault . . . . .	19-3
19.9 Floating Point Numerical Algorithms . . . . .	19-3
19.10 Eclipse Commercial Faults . . . . .	19-3
19.11 C/350 MMPU1 Instructions . . . . .	19-4

Data General Corporation  
Company Confidential

Chapter 20--C350 AND EAGLE PROGRAM COMBINATIONS . . . . .	20-1
20.1 Expanding An C350 User Program To Use 32 Bits ACs . . .	20-1
20.2 EAGLE Program Calling C350 Subroutines . . . . .	20-2
Chapter 21--C350 Map Compatibility . . . . .	21-1
21.1 Eagle Instructions . . . . .	21-1
Appendix I--Instruction Set Index . . . . .	I-1
Appendix II--Instruction Binary Encodings . . . . .	II-1
Appendix III--Scientific Instruction Set . . . . .	III-1

## Figure Table of Contents

Fig. # . . . . .	Page #
0	
2-1	Floating Point Format . . . . . 2-2
2-2	Fixed Point Format . . . . . 2-2
2-3	String Format . . . . . 2-3
2-4	Unpacked decimal, low-order sign/overpunch . . . . . 2-3
2-5	Unpacked decimal, high-order sign/overpunch . . . . . 2-3
2-6	Unpacked decimal, trailing sign . . . . . 2-4
2-7	Unpacked decimal, leading sign . . . . . 2-4
2-8	Unpacked decimal, unsigned . . . . . 2-4
2-9	Packed decimal . . . . . 2-4
2-10	Binary integer, signed . . . . . 2-4
2-11	Floating Point, byte aligned . . . . . 2-4
2-12	Commercial Eagle Descriptor . . . . . 2-5
3-1	Fixed Point Accumulator . . . . . 3-1
3-2	Program Counter . . . . . 3-2
3-3	Processor Status Register - PSR . . . . . 3-3
3-4	Floating Point Status Register . . . . . 3-4
3-5	Segment Base Register . . . . . 3-7
4-1	Word Pointer . . . . . 4-1
4-2	Byte Pointer . . . . . 4-2
4-3	Zero Extended 15 Bit Absolute Word Displacement . . . . . 4-3
4-4	Zero Extended 16 Bit Absolute Byte Displacement . . . . . 4-4
4-5	Bit Offset . . . . . 4-5
5-1	Word Logical Address - Memory Management . . . . . 5-2
5-2	Segment Base Register . . . . . 5-2
5-3	Page Table Entry . . . . . 5-4
5-4	1 Level Page Table - Logical Address . . . . . 5-5
5-5	2 Level Page Table - Logical Address . . . . . 5-5
5-6	One Level Page Table Translation . . . . . 5-7
5-7	Two Level Page Table Tr_
8	translation.- con't on next page 5-8
5-8	Two Level Page Table Translation . . . . . 5-9
6-1	Logical Address Space Structure . . . . . 6-2
6-2	Source/Target Address Space-Valid References . . . . . 6-3
6-3	Inward Call Branch Address . . . . . 6-8
6-4	GATE ARRAY . . . . . 6-9
8-1	Call Created Double Word . . . . . 8-2
8-2	Top of Stack Entry . . . . . 8-2
8-3	Inward Address . . . . . 8-3
8-4	Wide Return Block - WSAVR/S . . . . . 8-6
8-5	Wide Return Block - WSSVR/S . . . . . 8-8
8-6	Wide Return Block - Before/After Popping . . . . . 8-10
8-7	Wide Restore Block - WRSTR . . . . . 8-13
9-1	Queue Descriptor - Empty Queue . . . . . 9-2

14:58:1  
21/Aug/80

Data General Corporation  
Company Confidential

9-2	Queue with one Data Element . . . . .	9-2
9-3	Queue with two Data Elements . . . . .	9-3
9-4	Queue with three Data Elements . . . . .	9-4
13-1	IOSB Command - Format 1 (C I/O Format) . . . . .	13-6
13-2	IOSB Command - Format 0 (P I/O Format) . . . . .	13-6
13-3	IOC Register 7700<8> . . . . .	13-6
13-4	IOC Register - 7701<8> . . . . .	13-7
14-1	Vector Table . . . . .	14-6
14-2	DCT Entry - First 5 Words . . . . .	14-6

Data General Corporation  
Company Confidential

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



## Preface

This revision, Rev. 7, defines the software visible architecture of the ECLIPSE MV/8000, the first incarnation of the Eagle architecture. Additionally, an extensive set of array and intrinsic instruction primitives are included. These primitives are NOT supported in the MV/8000.

Relative to the Rev. 6 of the architecture spec, the following changes are incorporated in this document. All significant changes are denoted by revision bars in the left hand margin.

- 1) The actions taken when a protection fault occurs during the execution of the CALL and WPOPB instructions.
- 2) The elimination of the page residency checks from the LSBR instructions. The particular implementation's page fault control sequences are now responsible for detection of this error.
- 3) The OR and RESET referenced bits instructions are changed.
- 4) The state of the IO protect bits (per rev. 6.1)
- 5) Commercial faults are modified (per. rev. 6.1)
- 6) Floating Point synchronization with the instruction stream, when an asynchronize floating point unit exists in an  
im\_ \_\_\_\_\_plementation.
- 7) New instructions per C. Holland's memo dated 7/9/80. An extensive list, generally fixed point involving the use of literals.
- 8) Fixed point overflow, or lack of it thereof when OVR and OVK are simultaneously loaded.
- 9) An unimplemented opcode feature.
- 10) The extended array and instrinsic instruction set option. Specified in appendix III. (SIS.LS by Doug Geller)

--End of Preface--

10:3:27  
17/Sep/80

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 1 Overview

Nothing in this world is so powerful as an  
idea whose time has come.

Victor Hugo

### 1.1 Introduction

This document is a specification of the EAGLE architecture. An architectural specification describes the software visible aspects of a computer system. As such, there is no implementation implied. Any computer system adhering to the functionality described within will be capable of executing a common set of software. This feature is highly desirable.

The primary objectives of Eagle are: to provide present Eclipse users an orderly growth path to a large logical address space, while retaining their present software investment; and to allow new applications to use the vast amount of Eclipse system software. To accomplish this, the Eagle architecture is a superset of the Eclipse. That\_

\_\_\_\_\_P\_\_\_\_\_ is, all Eclipse C350 instructions exist in Eagle. In addition to the Eclipse instructions, functionality is added to assist in the manipulation of an expanded logical space.

The chapter summarizes the attributes of EAGLE, and introduces terms used throughout this document. Subsequent chapters detail the EAGLE architecture.

### 1.2 EAGLE Architecture Summary

The logical address space of EAGLE is 4.3 billion bytes (a byte has 8 bits of precision), requiring a 32 bit logical address field. Within EAGLE, this logical address can take three forms. One form is a byte address. This requires a 32 bit logical address. Another form is a word address. (A word operand has 16 bits of precision.) A word logical address requires a 31 bit logical address. The last form is a bit address. A bit logical

Data General Corporation  
Company Confidential

Rev. 7

address has a word address for a base and a bit offset.

The following operand types are supported by EAGLE:

- \* Bytes
- \* Byte Strings
- \* Bits
- \* 16 bit fixed point integers (word)
- \* 16 bit logical operands
- \* 32 bit fixed point integers (double word)
- \* 32 bit logical operands
- \* 32 bit floating point (single precision)
- \* 64 bit floating point (double precision)
- \* Eclipse commercial data types

In Eagle the programmer visible register set has been expanded. There are, as X in the Eclipse, 4 fixed point accumulators, and a program counter. These registers have been extended to 32 bits of precision. Management of the stack associated with the expanded logical address is aided by the inclusion of 4 32\_bit registers: the stack pointer, stack limit, stack base, and frame pointer. The floating point status register is expanded to 64 bits. There are several new registers associated with memory management.

The interrupt and fault mechanisms have been augmented to allow for saving state when expanded logical addresses are being used.

The Eagle I/O structures are totally compatible with the Eclipse. Eclipse I/O instructions exist in Eagle. All Data Channel and Burst Multiplexor Controllers are supported in the Eagle Architecture.

Eagle contains a protection mechanism based on Rings. Rings define an hierarchical access control mechanism that allows the protection of system software from unmediated access by a user. Rings are directly supported by the Eagle addressing mechanism.

Data General Corporation  
Company Confidential

Rev. 7

The memory management mechanism supports a demand paged environment. As a result of the large logical address space, memory management is handled in a manner differently from the present Eclipse map.

### 1.3 EAGLE/ECLIPSE Relationship

As previously stated, EAGLE is a superset of the Eclipse architecture\_

\_\_\_\_\_. The Eclipse architecture used as the base for the EAGLE superset is the C350. The C350 instruction set is basically an M600 without demand paging. Thus, the demand paging facilities on the M600 are not supported. To define the superset without the use of a "mode" bit, certain C350 capabilities were not supported. These capabilities; auto-increment and auto-decrement, and the no-load always skip instructions, were used as escape mechanisms to define EAGLE. A later chapter details these escape mechanisms and their effect on C350 compatibility. Subject to the above caveats, C350 application and system programs run without modification.

Eagle programs that use the expanded logical address space can not reliably contain C350 memory reference instructions. Conversely, to take advantage of the expanded logical address space, C350 programs must have all memory reference instructions replaced with expanded address instructions.

Throughout the remainder of this document the term Eagle means instructions or capabilities that use expanded addressing. The term Eclipse or C350 means instructions or capabilities that use 64kb addressing.

The term present or current segment, and present or current ring means the segment or ring implied by bits 1-3 of the present value of the Program Counter.

Any field marked as RESERVED should be considered as hardware reserved unless otherwise specified. This field should be initialized to zero. On an implementation specific basis, there may be a hardware check to verify that the reserved fields are in fact 0.

The symbol "\*" denotes multiplication. The symbol "\*\*\*" denotes exponentiation.

--End of Chapter--

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



## Chapter 2 Data Types and Formats

### 2.1 Basic Allowable Types

In this architecture, a double word has 4 bytes; a word has 2; a character, one. Double precision refers to 2-double word quantities; single precision refers to 1-double word quantity. This chapter enumerates the data types supported by the architecture, and describes the formats they take while residing in system memory.

There is no architecturally imposed memory alignment for the data types listed in this chapter. Operands referenced by word addresses can begin on any word boundary. Operands referenced by byte addresses can begin on any byte boundary. Certain instructions may require that the referenced operand begin on an integral boundary. Such exceptions are noted with the instruction definition.

The following formats can be found in the reference manual, "Eclipse C/350 Principles of Operation", manual number 014-000610-00

#### 2.1.1 Floating Point

Real numbers are represented in standard Data General (and IBM) format.

Data General Corporation  
Company Confidential

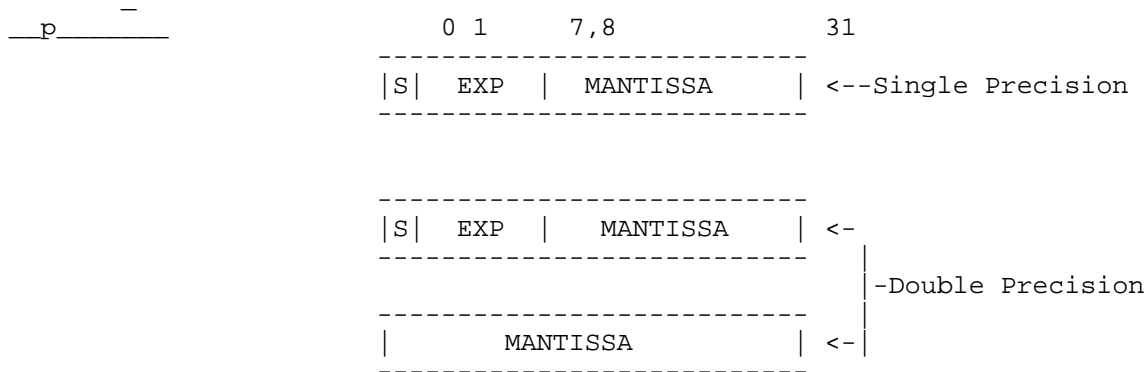


Figure 2-1. Floating Point Format

## 2.1.2 Fixed

Fixed point signed numbers are supported in 2's complement integer representation. Direct support is provided for 16- and 32-bit container sizes. The sign bit, bit 0, has the value  $-(2^{n-1})$ ; where n is the total number of bits contained in the fixed point signed integer.

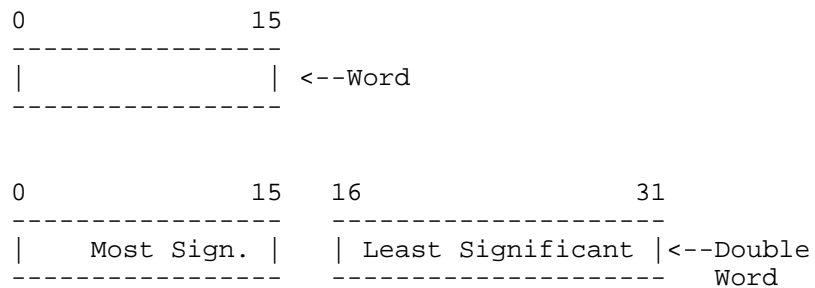


Figure 2-2. Fixed Point Format

## 2.1.3 Boolean

Boolean values occupy a one bit container and have the value zero or one.

Data General Corporation  
Company Confidential

Rev. 7



Data General Corporation  
Company Confidential

Rev. 7

Figure 2-6. Unpacked decimal, trailing  
sign

Figure 2-7. Unpacked decimal, leading sign

Figure 2-8. Unpacked decimal, unsigned

Figure 2-9. Packed decimal

Figure 2-10. Binary integer, signed

Figure 2-11. Floating Point, b\_  
\_\_\_\_\_yte aligned

The formats are as follows:

A commercial descriptor is always associated with a commercial data type. The format of the commercial Eagle descriptor is:

Data General Corporation  
Company Confidential

Rev. 7



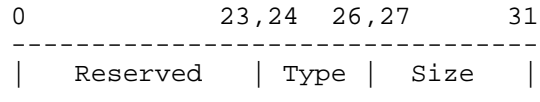


Figure 2-12. Commercial Eagle Descriptor

Bits	Name	Contents or Function
0-23	---	Reserved for future hardware use. Must be 0.
24-26	Type	Data Type: 0-Unpacked decimal, low order sign/overpunch 1-Unpacked decimal, high order sign/overpunch 2-Unpacked decimal, trailing sign 3-Unpacked decimal, leading sign 4-Unpacked decimal, unsigned 5-Packed decimal 6-Two's complement integer, byte aligned 7-Floating Point, byte aligned
27-31	Size	Data length: For all except data type 5, count of bytes in number minus 1 (including sign); For data type 5, the count of digits (nibbles) in the number(excluding the sign). The count is assumed to be odd. If a_
_____n even count is		specified, 1 is effectively added to make this count odd, and a zero digit is effectively appended to the most significant digit.

NOTE: Decimal Numeric Instructions do not necessarily use all these fields.

## 2.2 Floating Point Rounding

There are two rounding modes provided for floating point operations. Bit 8 of the Floating Point Status Register (FPSR) indicates the present rounding mode. The bit 8 encoding is:

Bit 8 = 0; Truncation - one guard digit is used in intermediate calculations. The intermediate result is TRUNCATED

Data General Corporation  
Company Confidential

Rev. 7

to obtain the final result.

Bit 8 = 1; Unbiased Round - two guard digits are used in intermediate calculations. The intermediate result is ROUNDED to obtain the final result.

Please see the floating point chapter for a complete description of these algorithms.

--End of Chapter--

Data General Corporation  
Company Confidential

This chapter describes the programmer visible Eagle register set and the page 0 locations in memory that have a pre-defined use. These topics will now be separately discussed.

### 3.1 Register Set

The register set is comprised of the following groups of registers: fixed point, floating point, stack management, and memory management.

#### 3.1.1 Fixed Point

There are 4 fixed point Eagle accumulators and one Eagle Processor Status Register (PSR). Each fixed point accumulator maintains 32 bits of precision. A fixed point accumulator can contain a fixed point integer or an address.

The structure of an accumulator is:

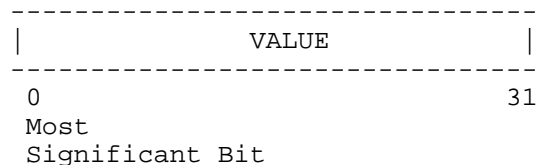


Figure 3-1. Fixed Point Accumulator

When a 16 bit fixed point integer or a 16 bit fixed point result is moved to an accumulator by an Eagle Instruction, the operand is sign extended to 32 bits prior to storage into the register. When a 15 bit address is moved to an accumulator, the address is first zero extended to 28 bits. Then the higher order 3 bits of the PC is appended to this 28 bit operand , and a zero is

Data General Corporation  
Company Confidential

Rev. 7

then appended to this \_  
 \_\_\_\_\_ 31 bit operand prior to storage into an  
 accumulator.

When a byte is moved to an accumulator by an Eagle instruction, it is zero extended to 32 bits prior to storage into the register.

The program counter is 31 bits long. The structure of the program counter is:

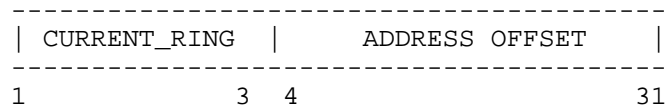


Figure 3-2. Program Counter

Bit 1-3 specify the current ring of execution. Bits 4-31 specify an instruction address. When the PC is incremented to reference the next sequential instruction, only the least significant 28 bits are used. Instructions exist which alter the current ring of execution.

The Eclipse 16 bit accumulators and 15 bit program counter are mapped onto the appropriate Eagle registers as follows. The Eclipse 16 bit accumulators are the least significant 16 bits of the Eagle accumulators. Generally the execution of Eclipse instructions leaves the most significant 16 bits of the Eagle accumulators in an undefined state. The Eclipse program flow instruction alter the Eagle PC in the following manner. Bits 1-3 of the Eagle PC remain unchanged. Bits 4-16 of the Eagle PC become all 0's. Bits 17-31 are loaded with the result of the execution of the Eclipse program flow instructions. Please see the instruction set chapter for a detailed explanation of the effect Eclipse instructions have on Eagle registers.

The structure of the Eagle Processor Status Register (PSR) is:

17/Sep/80

Rev. 7

Data General Corporation  
Company Confidential



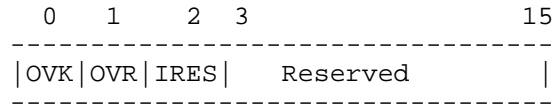


Figure 3-3. Processor Status Register - PSR

The meaning of bits 0-15 are:

BITS	NAME	MEANING
0	OVK	Overflow Mask. If this bit is set, the setting of bit 1 will result in a fixed point overflow.
1	OVR	Fixed Point Overflow Indicator.
2	IRES	Micro Interrupt Resume.
3-15		Hardware Reserved. Must be 0.

### 3.1.2 Floating Point Registers

There are 4 floating point accumulators (FPAC) and one floating point status register (FPSR). The floating point accumulators contain 64 bits of precision. This is sufficient to wholly contain a double precision floating point value. The Eagle and Eclipse floating point accumulators are identical.

The floating point status register contains 64 bits of precision. This register has the following structure:

Data General Corporation  
Company Confidential

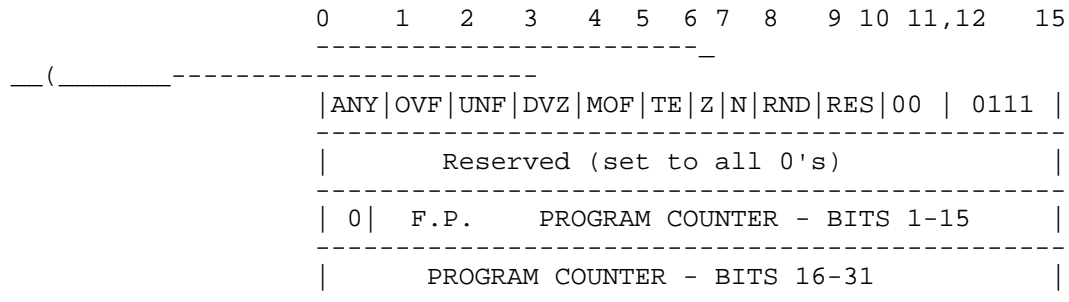


Figure 3-4. Floating Point Status Register

The meaning of bits 0-15 are as follows:

BITS	NAME	CONTENTS or FUNCTION
0	ANY	Indicates that any of bits 1-4 are set.
1	OVF	Exponent Overflow Indicator.
2	UNF	Exponent Underflow Indicator.
3	DVZ	Divide by zero.
4	MOF	Mantissa overflow during FSCALE or FIX instructions.
5	TE	Trap Enable. If this bit is set, setting any of bits 1-4 will result in a floating point fault.
6	Z	Zero bit.
7	N	Negative bit.
8	RND	Floating Point Rounding Mode.
9	RES	Interrupt Resume.
10-11		Hardware Reserved. These bits must be zero. When the array processing instruction set is installed these bits are used for additional control of floating point overflow and underflow. Please see the appropriate appendix for a description of the meaning of these bits.
12-15	FPMOD	Floating Point Model. All Eagle Processors have a FP model of "0111".

See the Micro-Interrupt section in the Interrupt Chapter for a complete description of the setting of bit 9, \_

\_0\_\_\_\_\_ RES.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

Bits 33-63 are the Floating Point Program Counter. In the event of a floating point fault, this is the address of the first floating point instruction that caused the fault.

### 3.1.3 Stack Management

There are 4\_32 bit registers to manage the EAGLE stack. The registers are:

- \* WSP - Wide Stack Pointer
- \* WSL - Wide Stack Limit
- \* WSB - Wide Stack Base
- \* WFP - Wide Frame Pointer

The Eagle Stack Pointer references the double word entry at the top of the stack. A push operation increments all 32 bits of the stack pointer by 2 and then places the "pushed" object in the double word addressed by the new value of the stack pointer. A pop operation takes the double word addressed by the current value of the stack pointer and places it in some register and then decrements all 32 bits of the stack pointer by 2.

The Eagle frame pointer references the first available double word minus 2 in the current frame. The Eagle stack limit is an address that is used to determine stack overflow. After any stack operation that pushes objects onto the stack, the stack pointer is compared to the stack limit. If the stack pointer is greater than the stack limit, a stack fault is signalled. The Eagle stack base is an address that is used to determine stack underflow. After any stack operation that pops objects from the stack, the stack pointer\_

8

is compared to the stack base. If the stack pointer is less than the stack base, a stack fault is signalled.

The Eagle stack base and limit are initialized upon a cross ring call or by instructions which explicitly manipulate them. Instructions are defined which can explicitly use and manipulate ESP and EFP.

To be meaningful, the stack limit must be 24 to 32 words lower than the last word in the stack, because stack overflow is detected only at the end of a push operation (except in the case of the

Data General Corporation  
Company Confidential

Rev. 7

WSAVR/S or WSSVR/S instructions and copying arguments upon a cross ring CALL). Thus it is possible to push a 12 to 20 word return block starting at the stack limit. Stack overflow would not be signalled until the last double word of the return block is pushed. After the last double word is pushed, stack overflow is signalled, and another 12 word return block is pushed onto the stack by the stack fault mechanism. Depending upon the size of the initial return block (12 words for fixed point status and 20 words for floating point status), the potential overflow can be 24 to 32 words long.

The stack registers for a given segment are stored in one of two places. For the current segment they are stored in the hardware registers. For all other segments they are stored in page 0 locations 20-27<octal> of that segment.

Reference to the stack registers o\_  
 \_\_\_\_@\_\_\_\_\_f the current segment must  
 be made by the appropriate stack register instructions (see the section on Stack Control Instructions in the instruction chapter). Reference to the stack registers of all other segments must be made with memory reference instructions to locations 20-27 (octal) of that segment.

A program must not reference locations 20-27 of its segment.

The initial values of the Eagle Stack Management registers should be even. That is they reference memory locations that are aligned on a double word boundary.

Stack operations will still function if the alignment is on an odd word boundary. However, on an implementation specific basis, a performance degradation may occur for instructions which directly manipulate the stack.

### 3.1.4 Memory Management

There are 8 registers used to manage memory for a running Eagle process. These registers, SBRs (segment base register), have 32 bits of precision. The structure of a SBR is:

Data General Corporation  
Company Confidential

Rev. 7



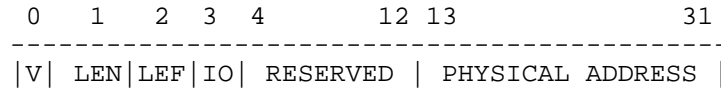


Figure 3-5. Segment Base Register

Where:

Bit 0 - Valid indicates whether or not the segment can be referenced. The encoding "0" means invalid and the encoding "1" means valid.

Bit 1 - Length - indicates the maximum length of the segment offset field. The encoding "0" denotes a 1 level pagetable. The encoding "1" denotes a 2 level pagetable.

Bit 2 - LEF - indicates whether an Eclipse LEF or an Eclipse I/O instruction is executed. The encoding "1" means a LEF is executed. The encoding "0" means an I/O instruction is executed.

Bit 3 - I/O Protection - indicates whether an I/O Protection Violation is signalled if an attempt is made to execute an I/O instruction. The encoding 1 implies execute the I/O instruction. The encoding 0 implies signal an I/O Protection Violation (code=10 in AC1).

Bit 4 - Reserved for hardware use.

Bits 5/12 - Reserved for software use.

Bits 13/31 - Physical Address - indicates the physical address in memory of the indicated page table.

See the chapter on memory management for a complete discussion of the use of these registers.

### 3.2 Page 0 Memory Locations

Data General Corporation  
Company Confidential

Rev. 7

Each segment has a "PAGE 0" (the first 256 words). Locations 0 thru 47 (octal) of page 0 are reserved for use by the system and the processor. There are two interpretations of the locations in page 0 of a segment. If the segment is 0, there is one interpretation. If \_

—P\_\_\_\_\_the segment is 1,2,3,4,5,6, or 7 (not segment 0) there is another interpretation. The interpretation of segment 0's page 0 is a superset of the interpretation of non-segment 0's page 0. Each of these interpretations are separately presented.

### 3.2.1 Segment 0 - Page 0

The meaning of segment 0's page zero locations are:

When MMPU1 (the C350's MAP) is enabled, locations 0,1,2, and 3 are physical. Locations 4,5,6, and 7 can be either physical or logical. When the Eagle ATU is enabled, all locations are logical.

Data General Corporation  
Company Confidential

Rev. 7

Word Location	Name	Function
0	I/O Return Address	Return Address from I/O interrupts. Eclipse Programs.
1	I/O Handler Address	Address of the I/O Interrupt Handler. Indirectable
2	SC Handler Address	Address of the Eclipse System Call Instruction. Indirectable
3 __X____ult	Protection Fa_ Address of the Eclipse Handler Address.	Protection Fault Handler. Indirectable.

When an Eagle Instruction is encountered as the first instruction of the interrupt handler; locations 0,1,2, and 3 have the following meaning. See the Interrupt Chapter for a detailed discussion.

0	Interrupt_Level	Level of interrupt Processing. 0 - Base. Not 0 - Intermediate.
1	I/O Handler	Address of the I/O Interrupt Handler. Indirectable.
2	I/O Return Address Eagle Higher Bits	Higher order bits of the Eagle I/O Interrupt Return
3	I/O Return Address Eagle Lower Bits	Lower order bits of the Eagle I/O Interrupt Return
4	Vector Stack Pointer	16 bit word offset. Lower order 16 bits of the Vector Stack:Pointer, Base, and Frame. Higher order 16 bits are set to 0.

Rev. 7

Data General Corporation

Company Confidential

\_\_\_\_, \_\_\_\_\_ -

5	Current Mask	Eclipse only. See the interrupt chapter for the Eagle mask.
6	Vector Stack Limit	16 bit word offset. Lower order 16 bits of the Vector Stack Limit.
7	Vector Stack Fault Address	Address of the Vector Stack Fault Handler. Indirectable
10-11	Eagle Breakpoint Address	Address of the Breakpoint Handler. Indirectable
12-13	Eagle XOP Origin Address	Address of the beginning of the Eagle XOP table. Non-indirectable.
14	Eagle Stack Fault Address	Address of the Eagle Stack Fault Handler. Indirectable
15	UIT Trap Address	Address of the UIT trap table in the current ring.
16-17	Reserved	Reserved

The following 8 page 0 locations are used for state saving and restoring upon Ring Crossing.

20-21	WFP	Eagle Frame Pointer
<u>h</u>		Non-indirectable.
22-23	WSP	Eagle Stack Pointer Non-indirectable.
24-25	WSL	Eagle Stack Limit Non-indirectable.
26-27	WSB	Eagle Stack Base. Non-indirectable.
<hr/>		
30-31	Eagle Page Fault Handler	Address of the Eagle Page Fault Handler Indirectable.

Data General Corporation  
Company Confidential

Rev. 7



32-33	Eagle Context Block Pointer	Address of the base of the Context Block Save Area. Indirectable.
34-35	EGP	Eagle Gate Pointer. Address of the Gate Array. Non-indirectable.
36	Protection Fault Handler Address	Address of the Eagle
<u>p</u>	Protection Fault Handler.	Indirectable.
37	Fixed Point Fault Handler Address	Address of the Eagle Fixed Point Fault Handler. Indirectable.
40	Stack Pointer	Address of the Top of the Eclipse Stack Non-indirectable
41	Frame Pointer	Address of the Start of the Current Eclipse Frame Minus 1. Non-indirectable
42	Stack Limit	Address of the Last Normally Usable Location in the Eclipse Stack
43	Eclipse Stack Fault Address	Address of the Eclipse Stack Fault Handler Indirectable.
44	XOP Origin Address	Address of the Beginning of the Eclipse XOP Table
45	Floating Point Fault Address	Address of the Faulting Point Handler Indirectable.
46	Commercial Fault	Address of th_
<u>x</u> e	Address	Commercial Fault Handler. Indirectable
47	DERR trap	Trap handler for the DERR instruction.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

|

1

1

Data General Corporation  
Company Confidential

## 3.2.2 Segment 1,2,3,4,5,6,7 - Page 0

The meaning of segments 1,2,3,4,5,6, and 7 page zero locations are:

Word Location	Name	Function
0-7	Reserved	
10-11	Eagle Breakpoint Address	Address of the Breakpoint Handler. Indirectable.
12-13	Eagle XOP Origin Address	Address of the beginning of the Eagle XOP table. Non-indirectable.
14	Eagle Stack Fault Address	Address of the Eagle Stack Fault Handler. Indirectable.
15	UIT Trap Address	Address of the UIT trap table in the current ring.
16-17	Reserved	Reserved
The following 8 page 0 locations are used for state saving and restoring upon Ring Crossing.		
20-21	WFP	Eagle Frame Pointer. Non-indirectable.
22-23	WSP	Eagle Stack Pointer. Non-indirectable.
24-25	WSL	Eagle Stack Limit. Non-indirectable.
26-27	WSB	Eagle Stack Base. Non-indirectable.
30-31	Reserved	

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

32-33	Reserved	
34-35	EGP	Eagle Gate Pointer. Address of the Gate Array. Non-indirectable.
36	Reserved	
37	Fixed Point Fault Handler Address	Address of the Eagle Fixed Point Fault Handler. Indirectable.
40	Stack Pointer	Address of the Top of the Eclipse Stack Non-indirectable
41	Frame Pointer	Address of the Start of the Current Eclipse Frame Minus 1. Non-indirectable
42	Stack Limit	Address of the Last
—		Normally Usable Location in the Eclipse Stack
43	Eclipse Stack Fault Address	Address of the Eclipse Stack Fault Handler Indirectable.
44	XOP Origin Address	Address of the Beginning of the Eclipse XOP Table
45	Floating Point Fault Address	Address of the Faulting Point Handler Indirectable.
46	Commercial Fault Address	Address of the Commercial Fault Handler. Indirectable
47	DERR Trap	Trap handler for the DERR instruction.

Data General Corporation  
Company Confidential

Rev. 7

|



3.2.2

Segment 1,2,3,4,5,6,7 - Page 0

3-15

| | Indirectable

All Eagle page 0 locations are logical.

--End of Chapter--

10:3:27

17/Sep/80

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 4

### Logical Address Space

One must have a good memory to be able to keep the promises one makes.

"Human All Too Human"  
Friedrich W. Nietzsche

The logical address space of Eagle is a two dimensional segmented address space. There are 8 segments. Each segment contains  $2^{28}$ , 16 bit words (512 megabytes). Thus the total logical address space contains  $2^{31}$ , 16 bit words or 4.3 gigabytes (a byte has 8 bits of precision).

The structure of a logical address will be examined by considering a word pointer and then considering a byte pointer. The format of a word pointer is:

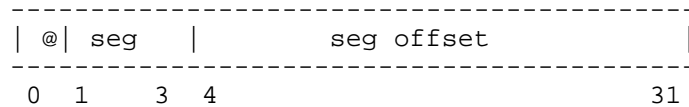


Figure 4-1. Word Pointer

where:

bit 0 indicates a presence ( a "1" encoding ) or absence ( a "0" encoding ) of a level of indirection.

bits 1-3 specify the segment number.

bits 4-31 specify the segment word offset.

The structure of a byte pointer is:

— \_\_\_\_\_l Corporation

Data Genera\_

Company Confidential

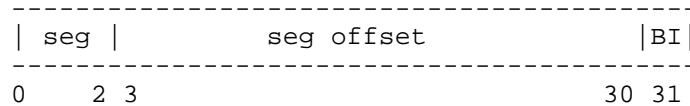


Figure 4-2. Byte Pointer

where:

bits 0-2 specify the segment number.

bits 3-30 specify the segment word offset.

bit 31 specifies the left byte (BI=0) or right byte (BI=1) of the reference word.

A byte pointer is encountered during the execution sequence of an Eagle instruction which specifically deals with bytes (e.g., Load Byte, Compare Strings.)

#### 4.1 Generation of a Word Logical Address

There are two addressing parameters that must be specified in order to generate a logical address. One parameter is the index and the other parameter is the length of the displacement field contained in the instruction.

The index specifies 4 addressing modes:

- 0 - Absolute
- 1 - PC based
- 2 - AC2 based
- 3 - AC3 based

When index 0 is specified the displacement field is used as an absolute address. If the displacement field specified in the instruction does not contain 31 bits of precision, then sufficient zero's are supplied to extend the specified displacement to a 31 bit absolute address within the current segment of execution. That is the 31 bit absolute address constructed has the form: bits 1-3 of the PC followed to the right by sufficient 0's catenated to the

\_\_\_\_\_  
17/Sep/80

Data General Corporation  
Company Confidential

Rev. 7

specified displacement field. Thus a 15 bit absolute displacement is extended to a 31 bit absolute address of the form:

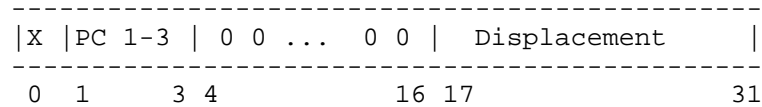


Figure 4-3. Zero Extended 15 Bit Absolute Word Displacement

When indexes 1,2, and 3 are specified, the displacement field is used as a 2's complement integer and algebraically added to the specified register to generate a logical address. If the specified displacement field does not contain 31 bits of precision, then a sign extension is performed to extend the displacement to 31 bits. For index 1 - PC based, the address of the word containing the 15 bit displacement or the address of the double word containing the 31 bit displacement is used as the specified register.

When an instruction is encountered (an Eclipse instruction) which manipulates a 15 bit word address, this address is extended to a 31 bit word address. The 31 bit address constructed has the form: bits 1-3 of the PC followed to the right by 13 0's catenated to the 15 bit word displacement.

## 4.2 Generation of a Byte Logical Address

There are two addressing parameters that must be specified in order to generate a byte logical address. One parameter is the index mode and the other parameter is the length of the displacement field contained in the instruction.

—0—

ment field contained in the instruction.

The index specifies 4 addressing modes:

- 0 - Absolute
- 1 - PC based
- 2 - AC2 based
- 3 - AC3 based

Data General Corporation  
Company Confidential

Rev. 7



When index 0 is specified the displacement field is used as an absolute address. If the displacement field specified in the instruction does not contain 32 bits of precision, then sufficient zero's are supplied to extend the specified displacement to a 32 bit absolute address within the current segment of execution. That is the 32 bit absolute address constructed has the form: bits 1-3 of the PC followed to the right by sufficient 0's catenated to the specified displacement field. Thus a 16 bit absolute displacement is extended to a 32 bit absolute address of the form:

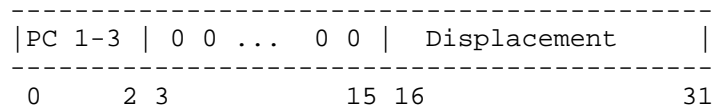


Figure 4-4. Zero Extended 16 Bit Absolute Byte Displacement

When indexes 1,2, and 3 are specified, the displacement field is used as a 2's complement integer and algebraically added to 2 times the specified register to generate a logical byte address. If the specified displacement field does not contain 32 bits of precision, then a sign extension is performed to extend the displacement to 32 bits. For index 1-PC based, the address of the word containing the 16 bit displacement or the address of the double word containing the 32 bit displacement is used as the specified register.

8. rmed to extend the displacement to 32 bits.

For index 1-PC based, the address of the word containing the 16 bit displacement or the address of the double word containing the 32 bit displacement is used as the specified register.

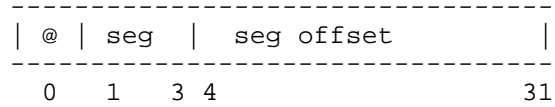
When an instruction is encountered (an Eclipse instruction) which manipulates a 16 bit byte address, this address is extended to a 32 bit byte address. The 32 bit address constructed has the form: bits 1-3 of the PC followed to the right by 13 0's catenated to the 16 bit byte displacement.

### 4.3 Generation of a Bit Logical Address

There are two parameters that must be specified in order to generate a bit logical address. One parameter is an indirectable word pointer. The other parameter is a bit offset. The word pointer is used as a base address of a field. The bit offset references a bit relative to this word base. The bit offset and word pointer are each contained in a Eagle fixed point accumulator. The format of a word pointer is:

Data General Corporation  
Company Confidential

Rev. 7



The format of the bit offset is:

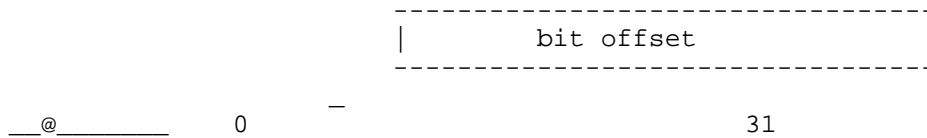


Figure 4-5. Bit Offset

When an instruction is encountered (an Eclipse Instruction) which manipulates bit addresses utilizing 15 bit word addresses, the address of the word containing the referenced bit is first resolved. This 15 bit word address is then expanded to a 31 bit word address. The desired bit in the word referenced by this 31 bit address is then manipulated.

If the ACD and ACS fields are identical in the Eagle Bit Instructions, then the word pointer is assumed to be zero in the current segment.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 5 Memory Management

I cannot be seen, cannot be felt,  
Cannot be heard, cannot be smelt.  
It lies behind stars and under hills,  
And empty holes it fills.

"The Hobbit"  
J.R.R. Tolkien

### 5.1 Introduction

Memory management is the process necessary to transform a logical address into a physical address and to determine what actually resides in physical memory. This process is achieved by:

1) Defining a standard memory allocation unit called a page. A page occupies 2048 (2KB) bytes of storage.

2) Defining a table (page of pagetable entries) which contains

\_\_\_\_\_H\_\_\_\_\_ains information necessary to ascertain the attributes of a referenced page.

3) Defining machine state to maintain flags that are used to manage the allocation of physical pages in memory.

### 5.2 Logical to Physical Translation

The logical to physical translation functions in the following manner. (In the following discussion the term referenced page means the page which contains the data addressed by the currently executing instruction. The term pagetable page denotes a 2 KB area of memory which contains pagetable entries. These entries are for memory management and protection. The term indicated page denotes either a referenced page or a pagetable page.)

The word logical address is partitioned into four fields for memory management purposes. The partitioning is:

Data General Corporation  
Company Confidential

Rev. 7

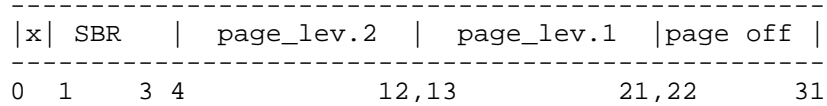


Figure 5-1. Word Logical Address - Memory Management

The segment field of a pointer (bits 1-3) references one of eight possible segment base registers (SBR). A SBR contains the following relevant memory management data:

1)The physical address of a pagetable page

2)The extent of the logical address defined for this

\_\_\_P\_\_\_\_\_ particular segment.

3)The validity of a reference to this segment.

The format of a SBR is:

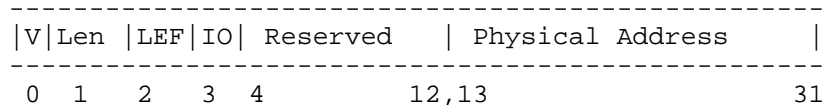


Figure 5-2. Segment Base Register

where:

bit 0 - Valid - indicates whether or not the segment can be referenced. 0 - invalid SBR, 1 - valid SBR

bit 1 - Length - indicates the maximum length of the segment offset field (28 bit word offset). The encoding denotes:

0 - 1 level page table. The maximum offset is 1 megabyte.

1 - 2 level page table. The maximum offset is 512 megabyte.

Data General Corporation  
Company Confidential

Rev. 7



bit 2 - LEF - indicates the interpretation of the Eclipse LEF instruction. If bit 2 is a "0", an I/O instruction is executed. If bit 2 is a "1", the Eclipse LEF instruction is executed.

bit 3 - I/O Protection - indicates whether an I/O Protection Violation is signalled if an attempt is made to execute an I/O instruction. The encoding 1 implies execute the I/O instruction. The encoding 0 implies signal an I/O Protection Violation (code=10 in AC1).

  X  

bit 4 - Reserved for hardware use.

bits 5/ 12 - reserved for software use.

bits 13 / 31 - Physical Address - indicates the higher order 19 bits of the physical address in memory of the indicated pagetable page. This address is catenated with 10 bits to form a new physical address. These 10 bits are constructed by appending a 0 to the least significant bit of the appropriate 9-bit page field used as an offset from the SBR. Thus a pagetable page must be on an integral 2 KB boundary. That is the least significant 10 bits of the physical address pointing to the first entry in the pagetable page must be zero.

### 5.3 Page Table Entry

The page fields of the segment offset are used to reference a pagetable entry (PTE). This entry is used to directly reference the addressed datum (if a one level pagetable), or directly reference another pagetable entry (if a two level pagetable). The format of a pagetable entry is:

Data General Corporation  
Company Confidential

Rev. 7

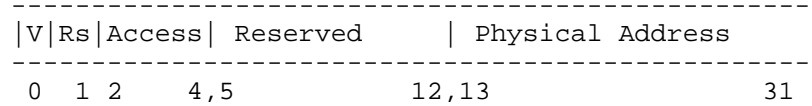


Figure 5-3. Page Table Entry

where:

bit 0 - Valid - indicates whether or not the indicated page has been defined. The encoding 0 implies invalid, and 1 implies valid.

bit 1 - Resident - indicates whether or not the indicated page is resident in main memory. If resident bits 13-31 specify the higher order 19 bits of a 29 bit physical word address. The encoding 0 implies non-resident, and 1 implies resident.

bits 2 / 4 - Access - explained in chapter on protection.

bit 5 - Reserved for hardware use.

bits 6/ 12 - Reserved for software use.

bits 13 / 31 - Physical Address - indicates the higher order 19 bits of the physical address in memory of a page. This address is catenated with the least significant 10 bits of a word logical address to form a word physical address. This word physical address is obtained in the following manner.

If a 1 level page table is used to translate logical addresses, the page\_level\_1 field was used as an index off of the SBR to obtain this PTE. The page offset field (bits 22-31) is catenated to the physical address in this PTE to form a word physical address of the referenced datum.

If a 2 level page table is used to translate logical addresses, then initially page\_level\_2 is used as an index off of the SBR to obtain a PTE. A 10 bit field is then constructed from page\_level\_1 catenated to the right with a 0. This 10 bit field is then catenated to the physical address in the PTE, to form a physical address of another PTE. This second PTE is obtained. The page offset field

\_\_h\_\_\_\_

17/Sep/80

Rev. 7

Data General Corporation  
Company Confidential

(bits 22-31) is then catenated to the physical address in this second PTE to form a word physical address of the referenced datum.

Consequently the structure of a logical address from the viewpoint of memory management is:

-----											
	X		SBR		Must be 0			page_lev.1		page off	
-----											
	0		1		3	4		12,13		21,22	31

Figure 5-4. 1 Level Page Table - Logical Address

-----												
	X		SBR		page_lev.2			page_lev.1			page off	
-----												
	0	1	3	4		12,13			21,22			31

Figure 5-5. 2 Level Page Table - Logical Address

When a 1 level page table is specified, bits 4-12 of a word logical address must be all 0. If not a page fault is signalled (page table depth fault).

During the logical to physical address translation procedure, the access bits in a page table entry referencing another page table are ignored. This can only occur for two level page tables. The following pages pictorially present the structure of the logical to physical address translation for a one level and for a two level page table.

There are two pointer structures; byte and word. When an address translation is performed on a word pointer, this pointer\_

\_\_\_\_\_p\_\_\_\_\_ is shifted left, zero filled, one position prior to the translation. This has the effect of converting a word pointer into a byte pointer. This conversion is only for the logical to physical

Data General Corporation  
Company Confidential

Rev. 7

address translation.

Data General Corporation  
Company Confidential



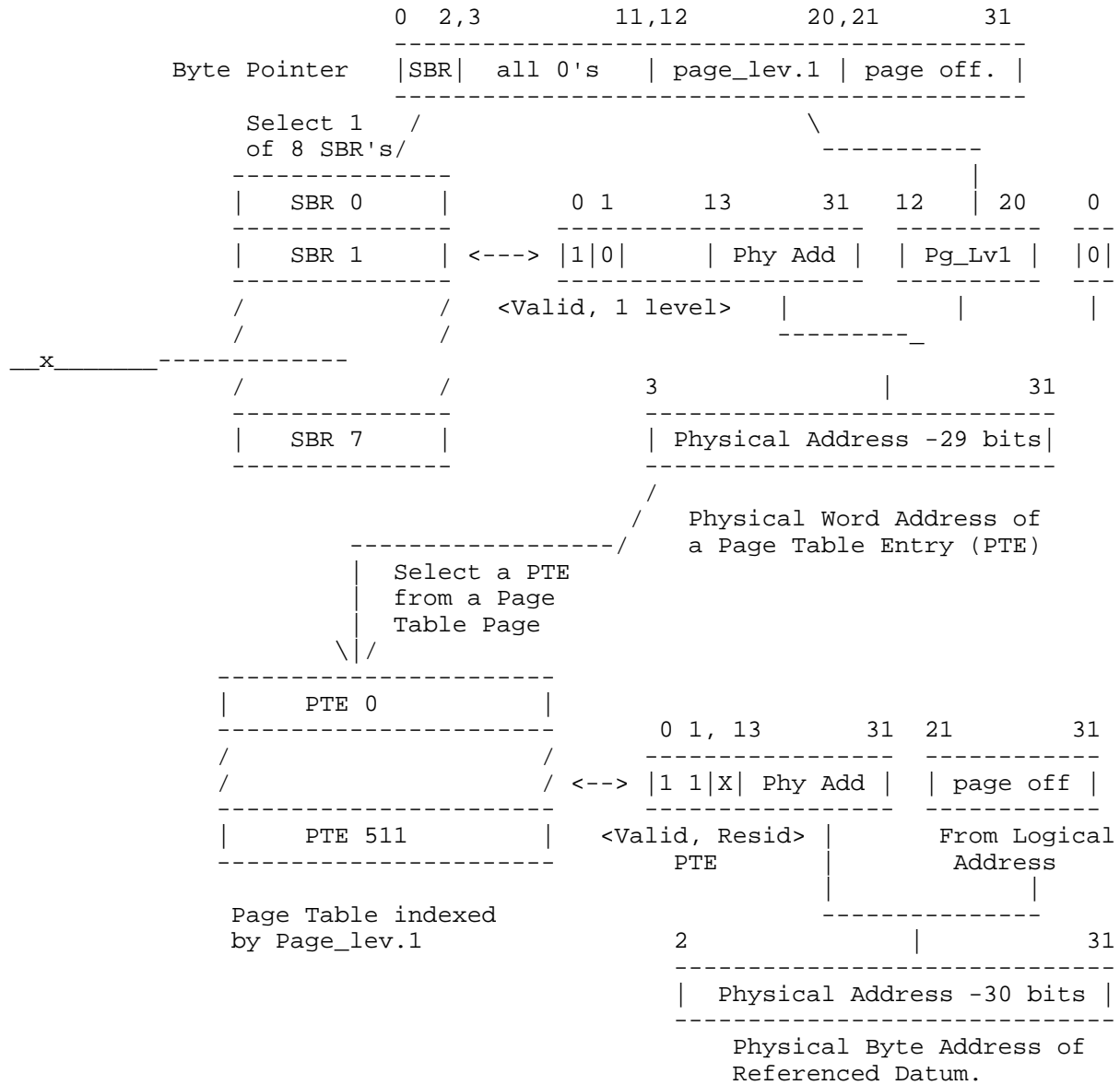


Figure 5-6. One Level Page Table Translation

— Data General Corporation  
Company Confidential

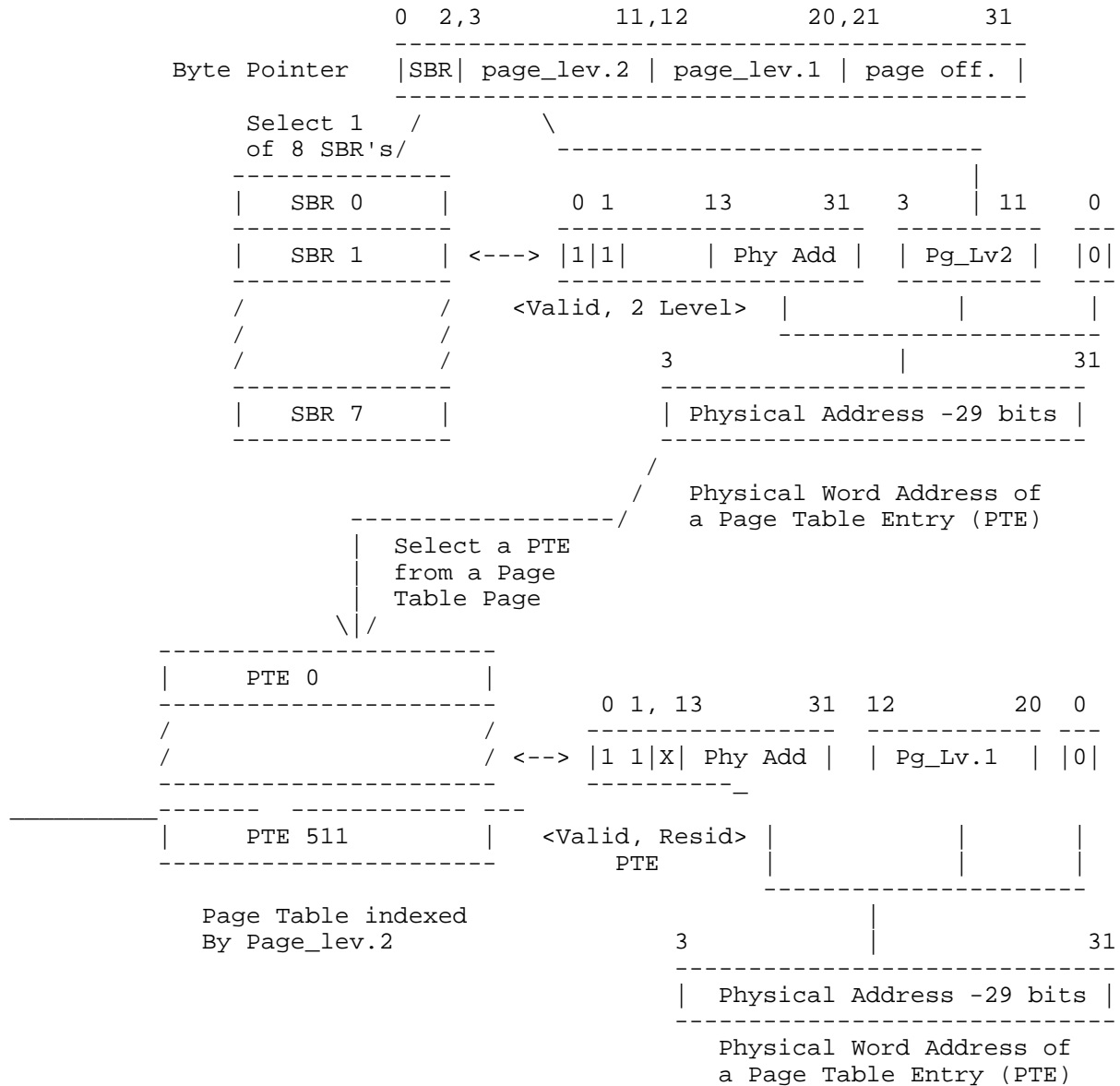


Figure 5-7. Two Level Page Table Translation.- con't on next page

Data General Corporation  
Company Confidential

Rev. 7

## TWO LEVEL PAGE TABLE TRANSLATION CONTINUED

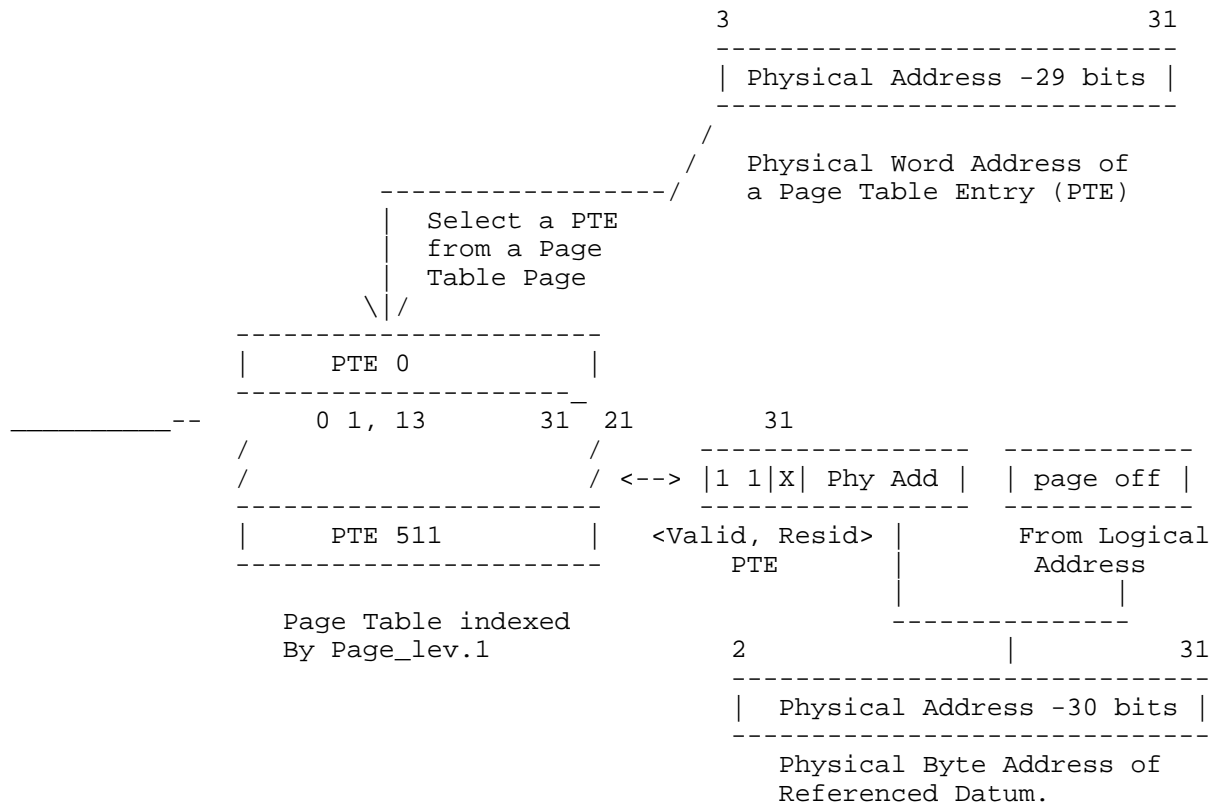


Figure 5-8. Two Level Page Table Translation

Data General Corporation  
Company Confidential

Rev. 7

#### 5.4 Physical Page Management

Associated with each physical page in memory are two flags: referenced and modified. When a successful read is initiated, the referenced flag associated with the physical page of the translated logical address is set to 1. When a successful write is initiated, the referenced and modified flags associated with the physical page of the translated logical address are both set to 1.

For the purposes of memory management, a successful action\_ is defined as a memory reference which does not result in a protection fault on a resident page.

The referenced and modified flags are set to 1 when a successful Processor reference is made. I/O memory references do not effect the state of these flags.

#### 5.5 Physical Addresses/ Power Up - I/O RESET

When power is first turned on, or after System Reset, virtual address translation is disabled. A logical address is equal to a physical address. The protection system functions as if Ring 0 is the current ring of execution. The state of the referenced and modified bits is indeterminate. The processor halts. The internal state of the processor is put in a restartable form. OVR ,OVK, and IRES (the PSR) are reset to 0. Instructions exist which cause the virtual address translation mechanism to be enabled and to restart the processor.

After the I/O Reset instruction is executed, virtual address translation is disabled. A logical address is equal to a physical address. The protection system functions as if Ring 0 is the current ring of execution. Additionally, bits 0,1, and 2 of the Processor Status Register (PSR) and bits 0,1,2,3,4,5,6,7,8 and 9 of the Floating Point Status Register (FPSR) are reset to zero after I/O reset. Instructions exist which cause the virtual address translation mechanism to be enabled.

When in physical address mode, Eagle effective address calculation functions in the same way as if virtual address translation were enabled. Since, the logical address space is greater than the physical address space, the most significant bits of the 31 bit logical address are truncated when presenting an address to physi-

17/Sep/80

Rev. 7

— —————  
Data General Corporation  
Company Confidential



cal memory. The number of bits truncated is implementation specific and is a function of the size of the physical address space.

When in physical address mode, executing an Eclipse LMP, SYNC, or MAP enable instructions function in the same manner as if they were executed on a C350.

## 5.6 Address Translation Unit (ATU)

In a previous section, the steps necessary to translate a logical address to physical address were presented. It is highly desirable once a translation is made, to remember the association between the logical and physical addresses. This desirability is a result of the following:

1) The steps necessary to translate logical addresses require processor cycles that would otherwise be allocated to the execution of instructions.

2) Programs exhibit both temporal and spatial locality of reference. Thus, it is highly probable that once a logical to physical translation is completed and remembered, subsequent logical addresses will reference the same page associated with the initial translation.

By remembering previous translations, the number of processor cycles allocated to logical address translations is significantly reduced. This feature is highly desirable. Program performance is greatly enhanced. Thus, an ATU is an accelerator.

The ATU accelerates address translations by associating a logical address with an entry. An ATU entry contains 3 types of

—(\_\_\_\_\_ information. One type is a physical address of a page.  
This

physical address is the contents of the PTE that referenced the addressed datum. Another type is the logical address associated with the physical address contained in the entry. Since the page offset field of a logical address is identical to the page offset field of a physical address, only the most significant 19 bits (at most) of the logical address are stored in an ATU entry. The last type of information is the access field contained in the PTE that referenced the addressed datum. Since the access privileges are on a per page basis, it is necessary to accelerate access privilege checking. The ATU entry associated with the logical to physical

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

translation provides a convenient place to store the access privileges of a page.

Some other characteristics of the ATU are:

- 1) The size and structure of the ATU is implementation specific.
- 2) Entries within the ATU are not explicitly addressable by the program.
- 3) Modifications of a PTE in memory does not necessarily have an immediate effect, if any, on ATU entries.

#### 5.6.1 Process Multiplexing

Now, that an ATU has been described, the dynamics of Process multiplexing and its affect on ATU entries are considered. A Process is defined as an abstraction of the locus of control that passes th\_

\_\_0\_\_\_\_rough an executing procedure.

Each Eagle Process has its own private logical address space. Thus, logical addresses though identical in two or more Processes need not translate to the same physical address. To eliminate the possibility of incorrectly associating a logical to physical translation belonging to one Process from being associated with a different Process, the ATU is purged. This purging occurs when a new Process is dispatched and begins executing on the processor. This purging mechanism simply marks all ATU entries as invalid. That is, no logical to physical associations exist in the ATU. Thus, no incorrect associations can result with identical logical addresses since no associations exist upon Process startup.

Due to the characteristics of Ring 0, it is often desirable to make Ring 0 procedure and data system wide (i.e., common to all Processes). Thus, under some circumstances only the non-Ring 0 entries of the ATU are purged.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 6 Protection System

One Ring to rule them all,  
One Ring to find them,  
One Ring to bring them all  
And in the darkness bind them.

"The Fellowship of the Rings"  
J.R.R. Tolkien

The protection system was specified using the following  
\_\_\_\_\_8\_\_\_\_\_ guidelines:

- 1)The operating system is part of the logical address space of the user.
- 2)The operating system is willing to perform some limited form of explicit checking to guard against trojan horse pointers ( the term will be explained).
- 3)The protection structure is not intended to provide controlled sharing among mutually suspicious subsystems.

The protection system is structured in the following manner. The logical address space is statically partitioned into eight hierarchical regions called RINGS. The partitioning is delineated by the segment field of the logical address. Segment number 0 is always assigned to Ring 0. Ring 0 contains the kernel. Privileged instructions can only be executed in Ring 0. Segment 1 is always assigned to Ring 1. And so on. This structure can be depicted as follows:

Data General Corporation  
Company Confidential

Rev. 7

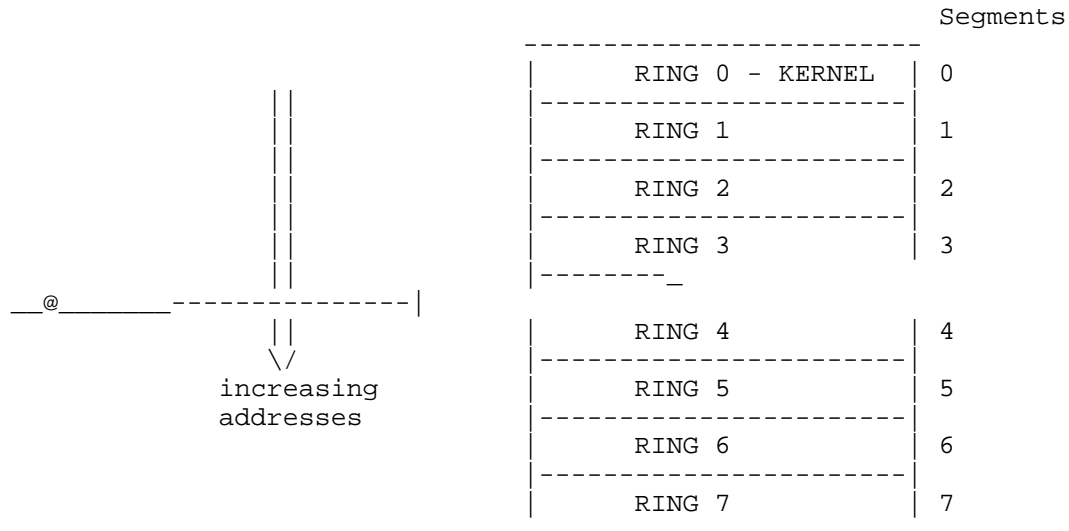


Figure 6-1. Logical Address Space Structure

This differs from classical segmented hierarchical address spaces in that the Ring # is not independent of the logical address space, but in fact directly bound in the space. In other words segment 0 is always allocated to RING 0.

Two additional structures are necessary to complete the protection system. One structure is the access field in a pagetable entry. This field (bits 2-4) indicates the capabilities of the referenced data item in the logical address space. The meaning of the access field of a valid pagetable entry is as follows:

- bit 2 - Read Access
- bit 3 - Write Access
- bit 4 - Execute access

The use of these bits will be subsequently described.

The other structure, ring maximization, governs the proper interpretation of the access privileges. The ring maximization function defines the conditions necessary for a reference to be valid. These conditions are defined by the following table (Table 1). In addition to the ring number bound to the address space, the access brackets are also bound to the address space.

Rev. 7

—  
H\_\_\_\_\_

Data General Corporation  
Company Confidential



An access bracket is a set of access privileges defined by consecutively numbered rings. If the effective source, as determined by ring maximization, is not within the appropriate access bracket, the reference is invalid. If the effective source is within the appropriate access bracket, the access field in the referenced page table entry is then used to determine the validity of the specific type of reference (i.e., read, write, or execute). In Eagle, the Read and Write Access Brackets for any particular segment number are Ring 0 up to and including the segment number. The Execute Access Bracket is limited to one ring and is equal to the segment number.

Thus, for example, the read bracket of a datum in ring 5 is 5, and so on. This means, that a data address reference to segment 5 can never legitimately originate from an effective source greater than 5 (for this example). If it does a protection fault is signalled (code=4 in AC1). The following table defines this address reference pattern. (A table entry details the new effective source and the validity of the reference)

Effective\ Source Space		Target Space				
		RING 0	RING 1	RING 2	...	RING 7
	RING 0	Val-R0	Val-R1	Val-R2	...	Val-R7
	RING 1	Fault	Val-R1	Val-R2	...	Val-R7
	RING 2	Fault	Fault	Val-R2	...	Val-R7
	.	.	.	—		
P	.	.	.	.		.
	.	.	.	.		.
	RING 7	Fault	Fault	Fault	...	Val-R7

Figure 6-2. Source/Target Address Space-Valid References

Where effective source space is initially indicated by the segment number of the program counter ( see subsequent section on indirection ). The target space is indicated by the segment number of the generated address.

Data General Corporation  
Company Confidential

Rev. 7

To determine whether or not a read, write, or execute access is allowed the following algorithms are defined:

1)Read Access - The ring maximization function is used to determine whether or not the reference is valid. If it is, bit 2 of the valid pagetable entry that references the addressed datum is examined. If bit 2 is a 1, the read is permitted. If a 0 or the PTE is invalid a protection fault is signalled (code=0 in AC1 for Read Access and code=3 for Invalid PTE).

2)Write Access - The ring maximization function is used to determine whether or not the reference is valid. If it is, bit 3 of the valid pagetable entry that references the addressed datum is examined. If bit 3 is a 1, the write is permitted. If a 0 or the PTE is invalid a protection fault is signalled (code=1 in AC1 for Write Access and code=3 for Invalid PTE).

3)Execute Access - If an intra-ring (within the same ri\_  
 \_X\_\_\_\_\_ng) transfer of control is performed, bit 4 of the valid pagetable entry is examined. If bit 4 is a 1, instruction execution is permitted. If bit 4 is a 0 or the PTE is invalid a protection fault is signalled (code=2 in AC1 for Execute Access and code=3 for Invalid PTE).

If an inter-ring transfer of control is performed, two forms of protection are enforced. The first form involves a valid Gate reference (see the section 7 of this chapter). The second form involves Execute Access validation of the page referenced by the Gate Array Entry.

## 6.1 Protection Notes

1) PC relative addresses are granted no special privileges. That is the appropriate read, write, and execute privileges as specified above apply.

2) The access privileges defined above are performed only if the pagetable entry associated with a valid address reference is valid. That is bit 0 of the pagetable entry must be a 1. The state of the resident bit is ignored for checking access privileges.

3) If an access privilege is changed for a process, after that process has already established a context in the ATU, the ATU must be PURGED upon completion of the alteration. ATU entries are not altered when a PTE is modified.

Data General Corporation  
Company Confidential

Rev. 7

4) If an instruction specifies an immediate operand (e.g., Add Immediate), the Read Access privilege of the page containing the immediate operand is not interpreted.

5) The source and target addresses are checked for ring

maximization validity for all instructions which produce effective addresses as the result (e.g., LLEF). This occurs, even though the final target address is not used by this instruction to obtain the referenced operand. Thus, for example, in the indirect chain of current ring of 4 referencing a pointer in ring 6, the ring field of the pointer is checked against 6. Similarly, when no indirection is specified, the effective address produced by evaluating the instruction displacement field and index register specification is compared to the source ring (in this case the current ring) for ring maximization validity.

6) The intermediate addresses of all instructions which can make multiple memory references (e.g., WCMV) are always ring maximized with the current ring to determine the validity of the intermediate reference.

7) The protection structure imposes a priority on the order protection faults are detected. This priority structure is (highest first):

- 0) Privileged Instruction.
- 1) Defer (levels of indirection).
- 2) Ring Maximization.
- 3) Valid SBR.
- 4) Valid PTE.
- 5) Appropriate Access check (i.e., Read, Write, or Execute).

Consequently, the occurrence of one protection fault renders a don't care all indications of lower priority protection violations.

8) During the logical to physical address translation procedure, the access bits in a page table entry referencing another page table are ignored. This can only occur for two level page tables.

## 6.2 Indirection

The protection structure defines a concept called effective source. Effective source functions in the following manner:

10:3:27  
17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

- 1) When a memory reference is made for the first time, the ring associated with the PC is the effective source.
- 2) Upon completion of a memory reference, the result of ring maximization becomes the new effective source.
- 3) Subsequent levels of indirection (if specified) follows rule 2.

Thus if a procedure in ring 0 indirections thru a pointer in ring 5, the first effective source is 0 (the ring of the PC). The reference to the pointer results in the new effective source becoming 5. (see table 1, this chapter). The new target space is indicated by the ring field of the indirect word.

### 6.3 SBR Validity Bit Protection

As previously described, if the validity bit in a PTE is a 0, a protection fault (code=3 in AC1) is signalled. A protection fault also occurs if the validity bit of the SBR referenced is a 0 (code=3 in AC1). In addition to these actions, the following occurs if the referenced SBR is SBR0. That is the SBR describing Ring 0.

If SBR0 is invalid, virtual translation is disabled and logical addresses are identical to physical addresses. The protection fault (due to the invalid SBR0) is now processed. The page 0 locations of segment 0 are in physical memory locations beginning with physical location 0.

### 6.4 Indirect Chain Protection

—  
—p———

A chain of 16 indirect references will cause a protection fault (code=5 in AC1). The indirection reference count is determined by the number of times indirection is specified (the number of "@" bits=1). Indirection protection is always enabled. If an instruction can resolve two or more pointers and thus two or more indirect chains, then the total number of indirect references for all pointer resolutions is a maximum of 16. WBLM is an example of this type of instruction.

Data General Corporation  
Company Confidential

Rev. 7



### 6.5 LEF Protection

Bit 2 of the SBR specified by bits 1-3 of the present value of the Program Counter controls the interpretation of the Eclipse "LEF" instruction. If bit 2 is a "1", the Eclipse LEF instruction is executed. If bit 2 is a "0", the Eclipse I/O instructions are executed.

### 6.6 I/O Protection

The ability to perform the execution of an I/O instruction is specified by bit 3 of the SBR of the current ring. When bit 3 is a "1", the fetched I/O instruction is executed. When bit 3 is a "0", an I/O Protection Violation (code=10 in AC1) is signalled.

### 6.7 Gate Access/Ring Crossing

In an hierarchical address space, it is desirable to mediate and authenticate any attempt to switch rings. In other words, ring switching must be performed in a well defined structure. The following delineates this structure.

#### 6.7.1 Initiation and Mediation of Ring Crossing

—  
 —x— Ring crossing can only occur as a result of an explicit attempt by a program control instruction. (It should be noted that when the PC is incremented to reference the next instruction, only the least significant 28 bits are incremented. Thus, logical address wraparound occurs within the current segment when the PC is incremented to reference the next instruction. )

An explicit attempt by a program control instruction can occur only if the following conditions are satisfied:

1) The program control instruction is of the form of a subroutine call or return (e.g., LCALL, WRTN, WPOPB). All other program control instructions ignore the ring field of the effective

Data General Corporation  
Company Confidential

Rev. 7

address. Thus, these instructions can only transfer to locations within the current segment.

2)The direction of the subroutine call crossing is inward, that is toward ring 0. Outward calls are trapped as a protection fault. The direction of the subroutine return is outward, that is away from ring 0. Inward returns are trapped as a protection fault (codes=7,8 in AC1 respectively).

3)The target segment of the effective branch address is not in the segment as indicated by bits 1-3 of the PC. If all the above conditions are met, the branch address, for inward calls, is interpreted as follows and the following actions occur.

NOTE: For outward returns the return address is interpreted as a normal word address.

-----										
X	SBR		NOT USED					GATE #		
-----										
0	1	3	4	15				16	31	

Figure 6-3. Inward Call Branch Address

Bits 16-31 are interpreted as a gate into the specified segment in the target space. To:

1)Verify that the specified gate is defined in the called segment.

2)Associate an instruction address with the specified gate, a GATE ARRAY is defined.

The location of the gate array is indicated by the pointer contained in logical locations 34 and 35 of the called segment. The structure of the gate array in this called segment is:

Data General Corporation  
Company Confidential

Rev. 7

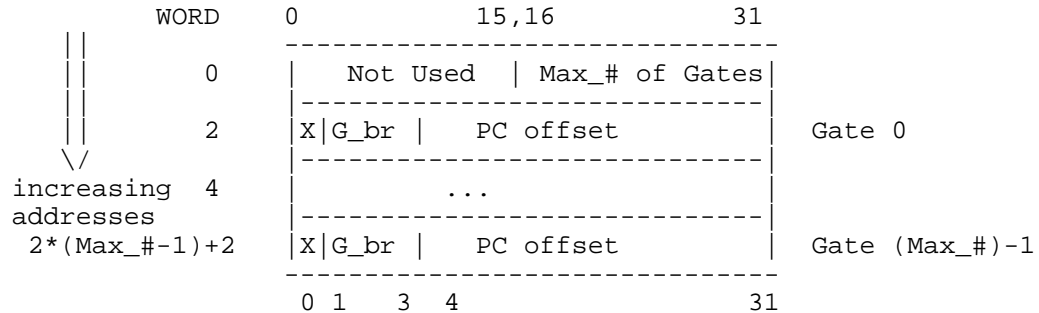


Figure 6-4. G\_

ATE ARRAY

The gate number of the pointer which referenced the target segment is compared with bits 16-31 of the first 32 bits of this gate array. If the gate number is less than the Maximum number of Gates then the gate number is used to index into the gate array which follows the max # of gates. The contents of the Gate indexed into is read and is used to control two actions. One action is a gate bracket compare. The effective source is compared to bits 1-3 of the referenced Gate. Both fields are interpreted as 3 bit unsigned integers. The effective source must be less than or equal to bits 1-3 of the Gate. If the comparison is true, the second action occurs. The PC offset (in the referenced Gate) becomes the least significant 28 bits of the program counter. Bits 1-3 of the PC are set to the segment containing the Gate Array. If the comparison is false or the gate number is greater than or equal to the max number of gates, the cross ring call is not permitted and a protection fault is signalled (code=6 in AC1). Thus, the protection fault occurs in the current ring. If the maximum number of gates is 0, this segment can not be a valid target of an inward ring crossing.

As a result of the Gate Bracket compare using the effective source, the indirect sequence (e.g.) current ring of 4 indirecting thru a pointer in ring 6, and the contents of this pointer indicating no further indirection and referencing ring 4, produces the following actions. The effective source is 6. Executing the X or LCALL instruction in ring 4, results in a Gate Bracket compare and Gate Index even though a Ring change does not occur. The Gate actions occur as a result of the indirect chain referencing a higher ring.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

## 6.7.2 Stack Switching and Argument Pushing

There is one stack per ring.

A cross ring call is initiated in the previously specified manner. The effective address specifies a GATE. If the GATE is an illegal entry point, the ring crossing does not occur, and a protection fault is signalled (code=6 in AC1). If the GATE is a legal entry point to this ring, a new stack is constructed and arguments from the caller's ring are copied onto the callee's stack. These actions are now separately discussed.

Before a new stack can be created, the present stack management registers must be saved. EFP and ESP are stored in locations 20-23 respectively of page 0 of the caller's ring. ESL and ESB are assumed to be identical to locations 24-27 respectively of page 0 of the caller's ring. The callee's stack is now created. Locations 22-27 of page 0 of the callee are loaded into ESP, ESL, and ESB respectively. This now completes the creation of the new ring's stack (EFP is subsequently initialized by the WSAV (R or S) instruction).

Before the actual copy occurs, a check is made to determine if a stack overflow would occur if all the arguments are copied. If it would, the arguments are not copied, and a stack fault is signalled. The ring crossing is permitted and it is in this ring, that the fault is processed. The program counter in the fault return block contains the address of  
 \_\_\_\_\_ the first instruction to be  
 executed (the contents of the legal GATE). AC1 is loaded with the value of 2 to indicate this type of stack fault. The ring crossing is allowed since the stack overflow could be due to the correct specification of the number of arguments, but the stack created did not have sufficient capacity to hold the copied arguments.

Arguments are now copied from the caller's stack onto the newly created callee's stack. The number of arguments to be copied is obtained by the argument\_count operand as specified by the X or LCALL instruction. The copying of arguments from the caller's stack onto the callee's stack functions in a manner such that the order of argument pushes is maintained. Thus EFP negative displacements into the callee's stack reference arguments in the same exact order as if a cross ring call had been an intra-ring call.

The copying of arguments from the caller's stack is relative to the top of the caller's stack. In other words ESP relative. For example, if 3 arguments are to be copied, the top 3 double

Data General Corporation  
Company Confidential

Rev. 7



words of the caller's stack (ESP relative) are copied to the newly created callee's stack. And, the callee's stack pointer is adjusted (+6) to reflect the 3 copied arguments.

After the arguments are copied, a double word is pushed onto the callee's stack. This double word contains the PSR, and the argument\_count. The instruction\_

\_\_\_\_\_ referenced by the PC field in the GATE is now executed. This instruction is normally a WSAVE. WSAV(R or S) pushes a return block onto the newly defined stack and loads EFP with the updated value of ESP. This now completes the creation of the new stack.

All of the above sequencing is due to the execution of a X or LCALL instruction (other than the sequences initiated by the WSAV(R or S) which specified an address in another (inward) ring.

### 6.7.3 Trojan Horse Pointers

Consider the following scenario. The user (ring 6) calls a subroutine in the utility space (ring 2). As part of the call sequence pointers are passed as arguments (call by reference). One of the passed pointers references a datum in the utility space rather than the user space. Malicious or not such an action results in the protection mechanism allowing references to the utility address space as governed by the utility's space access privileges (Note the previous table where source and target spaces are both ring 2). To protect against this occurrence the following functionality is provided:

1)An instruction is provided that checks to see that a pointer chain is valid. This instruction also performs a comparison between a supplied ring and the final effective address produced by the pointer chain validation procedure. The supplied ring must be greater than or equal to the produced ring.

2)Instructions which fetch data backwards (i.e., decreasing memory addresses) will perform a check to validate that the ring field of the source data does not change during instruction execution. Move strings backwards is an example of this type of instruction (Ref. Note 6 under the \_

\_\_\_\_\_(\_\_\_\_\_ Protec-  
tion Note Section).

--End of Chapter--

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

6.7.3

Trojan Horse Pointers

6-12

10:3:27  
17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

Chapter 7  
THE EAGLE INSTRUCTION SET

We still have judgement here,  
that we but teach Bloody Instructions,  
which, being taught,  
return to plague th' inventor.

"Macbeth"  
William Shakespeare

The EAGLE instruction set includes the entire C/350 instruction set, plus the additional instructions defined in this chapter.

When an Eclipse instruction is executed, the least significant 16 bits of the referenced registers are used in the specified operation. The result of this operation are returned to the least significant 16 bits of a register if the instruction specifies such as operation. The most significant 16 bits of the Eagle accumulators are in an undefined state at the completion of this operation.

There are three exceptions to this rule. One exception is that

—0— all Eclipse program flow and Effective Address Load instructions (e.g., JMP, RTN, and ELEF) alter the Eagle PC or accumulator in the following way. The most significant 16 bits remain as; current ring followed by 13 0's for the PC modification and the most significant 17 bits remain as; 0, current ring followed by 13 0's for the accumulator modification. The Eclipse instruction alters the least significant 15 bits. The second exception relates to the Eclipse ALC instructions. When an ALC instruction is executed which specifies the No-Load option (bit12=1), the Eagle accumulators remain unchanged. This No-Load option is particularly convenient to use when you want to test for some condition without destroying the contents of the destination accumulator. The third exception is, if the Eagle ATU is enabled, the execution of the Eclipse LMP, SYC, and MMPUL instructions is inhibited and a Privileged Instruction Violation (code=9 in AC1) is signalled.

The following rules hold, in addition to those previous specified, concerning Eagle and Eclipse instructions.

- 1) All Eclipse instructions which specify an accumulator as a source of information (CLM, DOA, etc.), but do not load that Eclipse accumulator, leave the entire 32 bit accumula-

Data General Corporation  
Company Confidential

Rev. 7

tor unchanged. In other words the entire referenced source accumulator remains unchanged.

- 2) Any Eclipse or Eag\_8\_\_\_\_\_le instruction which specifies that a word address (not to be confused with word pointer), ignores bit 0. In other words, the state of bit 0 has no affect on the execution of the specified instruction.
- 3) Furthermore, in generating word addresses, the sum of the referenced index register (PC, AC2, and AC3) and the displacement produces a 31 bit result. Thus bit 0 of the referenced index register has no affect on the value of the logical word address produced.

All Eclipse instructions execute according to the specification presented in "Eclipse C/350 Principles of Operation", 014-000610-00. Please see Appendix II for a listing of the Eclipse and Nova instructions supported by the Eagle Architecture.

When the Eclipse EXECUTE instruction is executed, the 16 least significant bits of the specified accumulator are interpreted. If these bits specify an Eagle instruction, then the Eagle instruction is executed.

When the C/350 MMPU1 (MAP) is enabled, an attempt to execute an Eagle instruction (i.e., all Eagle instructions, regardless of their binary encoding) results in an Eclipse I/O Protection violation. Thus bit 2 of the map status register is set to 1. The PC in the Eclipse narrow return block references the Eagle instruction.

When the Program Counter (PC) is incremented to point to the next sequential instruction, only the least significant 28 bits take part in this operation (modulo  $2^{**}28$ ). Thus, PC wraparound occurs within the current segment.

For a complete and concise listing of the format and binary encodings of every instruction listed in this and subsequent chapters, please see Appendix II. This append\_@\_\_\_\_\_ix is the most definitive presentation on the binary encodings of Eagle instructions.

Many of the instructions described in this section are Eagle forms of Eclipse instructions. Consequently, an in-depth presentation of the instruction's functionality is not provided. Please see the C/350 Principles of Operation for such a description.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



In the following sections, the state of the CARRY flag and OVERFLOW condition at the completion of an instruction is presented. Any undefined Eagle opcode is executed as a NOP. The term "SKIP" is used to mean that the next 16 bit word, following the instruction is skipped.

### 7.1 Fixed Point Overflow

A fixed point overflow occurs as the result of the loading of an accumulator with a computed two's complement value that is not an exact representation or a divide with a zero divisor. The occurrence of fixed point overflow is indicated by the setting of the OVR flag. The initiation of a fixed point overflow trap is indicated by an OVR flag of 1 and an OVK OF1. If OVK is a 0 an overflow trap is not initiated.

OVR and OVK are part of the processor state of a running Eagle program. They are saved and returned upon subroutine entry and exit.

Letting "OVERFLOW" indicate the value of an overflow condition for each Eagle Instruction (OVERFLOW is a level. It only has meaning during \_

H\_\_\_\_\_the execution of an instruction), the OVR flag is loaded as follows (where ".OR." denotes inclusive OR):

OVR <-- OVR .OR. OVERFLOW

The OVR flag if ever set to a 1, remains a 1 regardless of the new values of OVERFLOW generated by Eagle Instructions. This allows an efficient means to determine if an overflow trap would have occurred for a sequence of instructions.

OVR is explicitly altered, however, when the following actions occur: Interrupts, Faults, Power Up, I/O Reset, System Reset, and execution of the instructions - WPOPB, WRTN, LCALL, XCALL, WSSVR, WSSVS, WRSTR, and SPSR.

The following instructions load OVR and OVK as part of their execution; SPSR, XVCT, WPOPB, WDPOP, WRTN, WRSTR, and WSAVS. If the newly loaded values of OVR and OVK are BOTH 1, a fixed point trap IS NOT INITIATED. OVR and OVK remain unchanged after the execution of these instructions unless subsequent instructions are executed which directly alter OVK and OVR. A fixed point overflow

Data General Corporation  
Company Confidential

Rev. 7

|

| DOES NOT occur if an arithmetic instruction producing no OVERFLOW  
 | is executed when OVR and OVK are one. Fixed point overflow is  
 | initiated only when OVK and OVERFLOW are both one. OVR is ignored.

For all Eclipse instructions OVERFLOW equals 0, thereby leaving OVR unchanged.

## 7.2 Fixed Point Indexed Address Instructions

The EAGLE instruction set has an increased number\_  
 \_\_P\_\_\_\_\_ of indexed  
 address instructions. Indexed address instructions either load, store, or modify data in a location whose address computation is based on the index field, displacement and indirection bit contained within the instruction format. In these EAGLE instructions the displacement may be either 15 or 31 bits and the data referenced in memory may be 1, 2, or 4 bytes in length. The following mnemonic conventions are used to distinguish between instructions with similar semantics:

### Displacement length:

X	Extended displacement (1 word)
L	Long displacement (2 word)

### Data size:

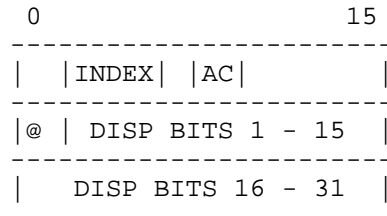
W	Wide (32 bits)
N	Narrow (16 bits)

The following instructions deal with 32 bit fixed point integers (W) and have 31 bit displacements (L). These instructions have two formats.

For all these instructions Carry <- Carry and OVERFLOW<-0

Data General Corporation  
Company Confidential

Rev. 7

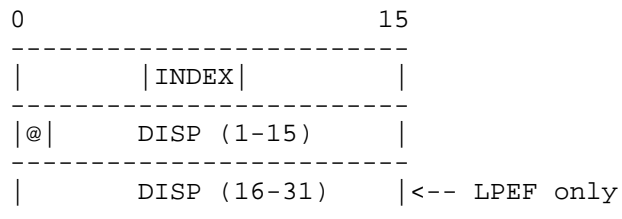


\* LLEF      Load a 31 bit effective address into an AC.

\* LWLDA    Load a 32 bit fixed po\_  
 \_\_X\_\_\_\_\_int integer into an AC.

\* LWSTA    Store a 32 bit AC to memory.

The following two instructions push an effective word address onto the stack. Carry <- Carry and Overflow <- 0. The format of these instructions are:



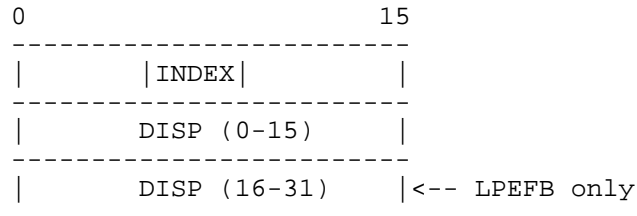
\* LPEF      Push the 31 bit effective word address onto the stack.

\* XPEF      Push the 31 bit effective word address onto the stack.

The following two instructions push a 32 bit effective byte address onto the stack. Carry <-- Carry and Overflow <-- 0. The format of these instructions are:

Data General Corporation  
Company Confidential

Rev. 7

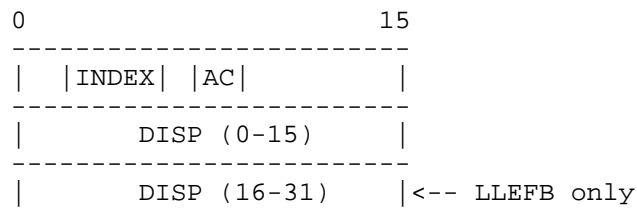


\* LPEFB    Push the 32 bit effective byte address onto the stack.

\* XPEF\_    Push the 32 bit effective byte address onto the stack.

\_\_\_`\_\_\_\_\_B

The following two instructions load a 32 bit effective byte address into the specified accumulator. Carry <-- Carry and Overflow <-- 0. The format of these instructions are:



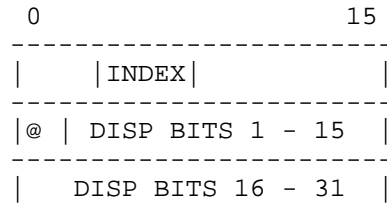
\* LLEFB    Load the 32 bit effective byte address into the specified accumulator.

\* XLEFB    Load the 32 bit effective byte address into the specified accumulator.

Data General Corporation  
Company Confidential

Rev. 7





The following two instructions are executed in one memory cycle (indivisible) if their word address is on a double word boundary. Carry←Carry and Overflow←0.

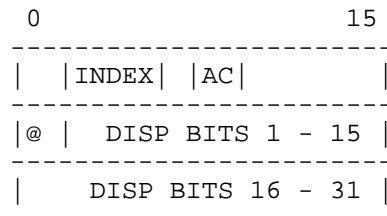
\* LWISZ Increment a 32 bit memory location and skip on

h — zero.

\* LWDSZ Decrement a 32 bit memory location and skip on zero.

The following instructions operate on 16 bit fixed point integers (N) and have 31 bit displacements (L). These instructions have two formats.

For all these instructions Carry ← Carry and OVERFLOW←0

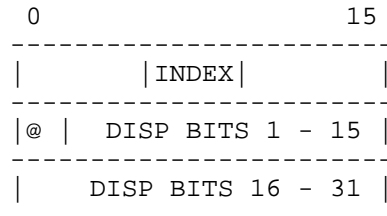


\* LNLDA Load a 16 bit fixed point integer into AC and sign extend to 32 bits.

\* LNSTA Store low order 16 bits of AC to memory.

Data General Corporation  
Company Confidential

Rev. 7



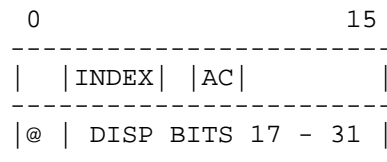
The following two instructions are executed in one memory cycle (indivisible). Carry<-Carry and Overflow<-0.

\* LNISZ Increment a 16 bit memory location and skip on zero.

\* LNDSZ           
p Decrement a 16 bit memory location and skip on zero.

The following instructions operate on 32 bit fixed point integers (W) and have 15 bit displacements (X). These instructions have two formats.

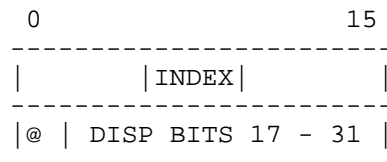
For all these instructions Carry <- Carry and OVERFLOW<-0



\* XWLDA Load a 32 bit datum into AC.

\* XWSTA Store a 32 bit AC to memory.

\* XLEF Load a 31 bit effective address into AC.



The following two instructions are executed in one memory cycle (indivisible) if their word address is on a double word

Data General Corporation  
Company Confidential

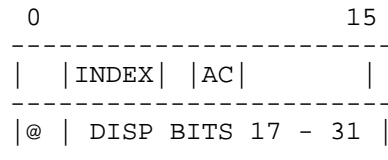
Rev. 7

boundary. Carry<-Carry and Overflow<-0.

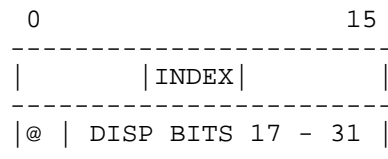
- \* XWISZ Increment a 32 bit memory location and skip on zero.
- \* XWDSZ Decrement a 32 bit memory location and skip on zero.

The following instructions deal with 16 bit fixed point integers (N) and have 15 bit displacements (X). These instructions are similar to C350 extended instructions, except that address calculation is 31 bits, allowing a short instruction to use 32 bit index registers and two word indirect locations. These instructions have two formats.

For all the following instructions, Carry<-Carry and OVERFLOW<-0.



- \* XNLDA Load a 16 bit datum into AC and sign extend to 32 bits.
- \* XNSTA Store a 16 bit AC to memory.



The following two instructions are executed in one memory cycle (indivisible). Carry<-Carry and Overflow<-0.

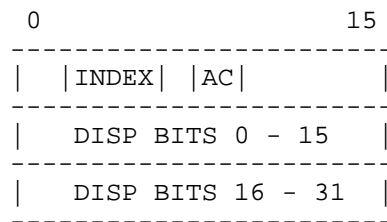
- \* XNISZ Increment a 16 bit memory location and skip on zero.
- \* XNDSZ Decrement a 16 bit memory location and skip on zero.

Data General Corporation  
Company Confidential

Rev. 7

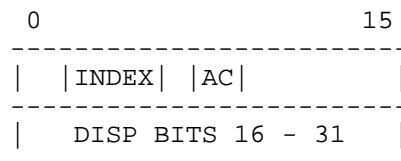
The following instructions operate on 8 bit bytes and have 32 bit byte displacement. The contents of the specified index register are i\_ \_\_\_\_\_nterpreted as a word address.

For all the following instructions, Carry<-Carry and OVERFLOW<-0.



- \* LLDB     Load a byte into AC with zero extend.
- \* LSTB     Store the low order byte into memory.

The following instructions operate on 8 bit bytes and have 16 bit byte displacements. The contents of the specified index register are interpreted as a word address.



- \* XLDB     Load a byte into AC with zero extend.
- \* XSTB     Store the low order byte of the AC into memory.

Data General Corporation  
Company Confidential

Rev. 7



## 7.3 Stack Control

The following instructions operate on the 32\_bit registers: stack pointer, stack limit, stack base, and frame pointer. They have the format:

For all these instructions Carry <- Carry and OVERFLOW<-0

0 15

-----  
AC

- \* LDASP            Load EAGLE stack pointer into AC.
- \* STASP            Store AC into EAGLE stack pointer.
- \* LDASL            Load EAGLE stack limit into AC.
- \* STASL            Store AC into EAGLE stack limit  
                    and into locations 24-25 of the current  
                    segment.
- \* LDASB            Load EAGLE stack base into AC.
- \* STASB            Store AC into EAGLE stack base  
                    and into locations 26-27 of the current  
                    segment.
- \* LDAFP            Load EAGLE frame pointer into AC.
- \* STAFP            Store AC into EAGLE frame pointer.

## WMSP - Wide Modify Stack Pointer

The specified AC is arithmetically shifted left one position and added to the Eagle Stack Pointer (ESP). The sum produced is temporarily saved in internal processor state. If a fixed point overflow is detected as a result of the left shift, ESP is not updated, and a stack fault is signalled (code=1 in AC1). Overflow is always 0 for this instruction. The detection of a fixed point overflow is treated as a stack fault. If the shifted AC is positive, a stack overflow check is performed using the sum temporarily saved. If the shifted AC is negative, a stack underflow check is performed using the sum temporarily saved. If an

\_\_\_\_\_n

Data General Corporatio\_  
Company Confidential

Rev. 7

underflow or overflow is not detected, ESP is loaded with the temporarily saved sum. If a stack overflow is detected, ESP is not updated, and a stack fault is signalled (code=1 in AC1). If a stack underflow is detected, ESP is not updated, and a stack fault is signalled (code=1 in AC1). The PC in the wide return block pushed references this instruction. Carry<--Carry and Overflow<--0

The following instructions allow efficient use of the ESP as a pointer to temporary storage. Carry<-Carry and Overflow<-0. These instructions have the following format: (except for ISZTS and DSZTS which have no accumulator specification)



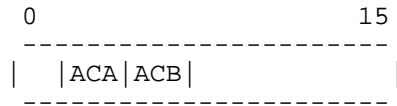
- \*  
LDATS    Load the double word pointed to by ESP into AC.
- \*  
STATS    Store the AC into the double word pointed to by ESP.
- \*  
ISZTS    Increment and skip on zero the double word pointed to by ESP (no AC specification). The operation performed by this instruction is not indivisible.
- \*  
DSZTS    Decrement and skip on zero the double word pointed to by ESP (no AC specification). The operation performed by this instruction is not indivisible.

The following instructions push or pop 32 bit fixed point accumulators from the EAGLE stack. These instructions specify two accumulators (ACA,ACB). As with the C350 instructions PSH and POP,

the accumulators are referenced in order from ACA to ACB inclusive. If ACB is less than ACA then they reference from ACA to AC3 followed by AC0 to ACB for push. For pop, if ACB is greater than ACA then they reference from ACA to AC0 followed by AC3 to ACB. For these instructions Carry <- Carry and Overflow <- 0. These instructions have the format:

Data General Corporation  
Company Confidential

Rev. 7



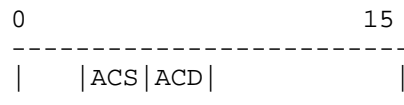
- \* WPSH     Push 32 bit accumulators on top of EAGLE stack.
- \* WPOP     Pop 32 bit accumulators from top of EAGLE stack.

Data General Corporation  
Company Confidential

## 7.4 Fixed Point Word Arithmetics

The following 16 bit arithmetics facilitate computation in which the detection of overflow is important. The 16 bit fixed point arithmetic instructions specify two accumulators: source (ACS) and destination (ACD). ACD is loaded with the sign extended 16 bit results of the least significant 16 bits\_

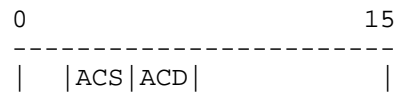
of ACD <OP> the least significant 16 bits of ACS. The most significant 16 bits of ACS and ACD are ignored during the formulation of the 16 bit intermediate result. These instructions have the formats:



- NADD    16 bit fixed point add. Carry <- ALU and OVERFLOW <- ALU.
- NSUB    16 bit fixed point subtract. Carry <- ALU and OVERFLOW <- ALU.
- NMUL    16 bit fixed point multiply. Carry <- Carry and OVERFLOW set if the product requires more than 16 bits of precision.
- NDIV    16 bit fixed point divide. The word integer contained in ACD(dividend) is divided by the word contained in ACS(divisor) and the 16 bit result (sign extended to 32 bits) is loaded into ACD. Carry <- Carry and OVERFLOW set if the divisor is zero or the most negative integer is divided by -1. If overflow is detected, the specified accumulators remain unchanged.

NNEG - Narrow Negate

The format of this instruction is:



The least significant 16 bits of ACS are negated. The negated 16 bits are then sign extended to 32 bits and loaded into ACD. The negation is performed as a two's complement subtract of ACS from 0. Carry <-ALU output.

— ( ————

—

Data General Corporation  
Company Confidential

Rev. 7



Overflow is set if an attempt is made to negate the largest negative 16 bit integer.

The following 16 bit arithmetic facilitate computation involving immediates. There are two formats: The 2 bit short immediate (N+1) and a 16 bit immediate. For all three of these instructions, OVERFLOW is set by the ALU output. Carry <- ALU output

- \* NADI     Add N+1 to the specified AC.
- \* NSBI     Subtract N+1 from the specified AC.
- \* NADDI    Add the 16 bit immediate to the AC.

Data General Corporation  
Company Confidential

## 7.5 Fixed Point Double Word Arithmetics

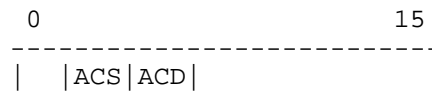
In order to facilitate 31 bit address computation, the EAGLE instruction set provides 32 bit fixed point arithmetic with the 32 bit fixed point accumulators.

The 32 bit fixed point add, subtract, increment, and negate (immediates and register to register) arithmetic instructions produce a carry out. The carry out is produced considering all 32 bits in the manipulation. The carry bit is set to carry out regardless of the initial value of the carry bit. Carry out is the carry from the most significant bit of the 32-bit arithmetic operation. All other 32 bit fixed point instructio\_

0\_\_\_\_\_ns leave the carry bit unchanged, unless otherwise specified.

OVERFLOW is set to the output of the ALU for Eagle Double Word fixed point arithmetics, unless otherwise specified.

The 32 bit fixed point arithmetic instructions specify two accumulators: source (ACS) and destination (ACD). ACD is loaded with the results of ACD op ACS. These instructions have the format:



WADD 32 bit fixed point add.

WSUB 32 bit fixed point subtract.

WADC 32 bit fixed point add complement of ACS to ACD.

WINC Increment 32 bit AC. Carry<-ALU output and Overflow<-ALU output.

WNEG 32 bit fixed point negative. Carry<-ALU output and Overflow<-ALU output.

WMOV 32 bit fixed point move. Carry<-Carry, OVERFLOW<-0

WXCH Exchange the 32 bit ACS with ACD. Carry<-Carry, OVERFLOW<-0

WHLV Halve a 32 bit AC. Carry<-Carry. One AC specification.

Data General Corporation  
Company Confidential

Rev. 7

WDIV     32 bit signed fixed point divide (dividend sign extended to 64 bits prior to the divide operation). The double word contained in ACD is divided by the double word contained in ACS and the double word quotient is loaded into ACD. Carry<-Carry. If Overflow is detect\_

\_\_8\_\_\_\_\_ed the specified accumu-  
lators remain unchanged. Overflow is set if the divisor is zero or the most negative integer is divided by -1.

WMUL     32 bit signed integer fixed point multiply. The result is the least significant 32 bits of the product. Carry<-Carry

SEX       Sign extend a 16 bit ACS to a 32 bit ACD. Carry<-Carry and Overflow<-0.

ZEX       Zero extend a 16 bit ACS to a 32 bit ACD. Carry<-Carry and Overflow<-0.

Data General Corporation  
Company Confidential

Rev. 7

## WASH - Wide Arithmetic Shift

Arithmetically shifts the contents of ACD either left or right depending on the number contained in bits 24-31 of ACS. The 8 bit two's complement number in ACS determines the direction of the shift and the number of bits to be shifted. If the number in bits 24-31 of ACS is positive, shifting is to the left and zeros fill vacated bit positions (integer multiply by a power of two). If the number in bits 24-31 of ACS is negative, shifting is to the right and the sign bit fills vacated bit positions (integer divide by a power of two). Right shifted numbers (positive and negative) are truncated (e.g., -3 shifted right one position yields -1; -1 shifted right one position yields 0). If the number in ACS is zero, no shifting is performed. Bits 0-23 of ACS are ign\_

@\_\_\_\_\_ored.

The carry bit and the contents of ACS remain unchanged. Fixed Point Overflow occurs if a left shift is specified and a shifted out bit is different than the value of the sign bit of ACD (Overflow<--1, otherwise Overflow<--0). Carry<--Carry If a fixed point overflow occurs, the contents of ACD are indeterminate.

## WASHI - Wide Arithmetic Shift Immediate

Arithmetically shifts the contents of ACD either left or right depending on the number contained in bits 24-31 of the immediate. The format of this instruction is:

0	15, 16	24	31
-----			
	ACD	Reserved	Count
-----			

The 8 bit two's complement number in the immediate field determines the direction of the shift and the number of bits to be shifted. If the number in bits 24-31 of the immediate is positive, shifting is to the left and zeros fill vacated bit positions (integer multiply by a power of two). If the number in bits 24-31 of the immediate is negative, shifting is to the right and the sign bit fills vacated bit positions (integer divide by a power of two). Right shifted numbers (positive and negative) are truncated (e.g., -3 shifted right one position yields -1; -1 shifted right one position yields 0). If the number in ACS is

—H—

Rev. 7

—

Data General Corporation  
Company Confidential

|

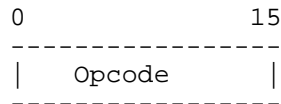


zero, no shifting is performed. Bits 16-23 of are reserved (must be zero).

Fixed Point Overflow occurs if a left shift is specified and a shifted out bit is different than the value of the sign bit of ACD (Overflow $\leftarrow$ -1, otherwise Overflow $\leftarrow$ -0). Carry $\leftarrow$ -Carry If a fixed point overflow occurs, the contents of ACD are indeterminate.

Data General Corporation  
Company Confidential

The following 2 instructions perform extended signed arithmetic to specific accumulators. The format of these two instructions is:

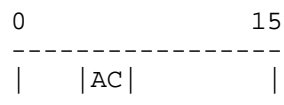


**WMULS** A 32 bit AC1 is multiplied by a 32 bit AC2 and added to the 32 bit AC0. The 64 bit result is placed in AC0 and AC1 with the high order bits in AC0. (signed AC's) Carry←-Carry and overflow←-0.

**WDIVS** A 64 bit signed number contained in AC0 and AC1 is divided by the 32 bit signed number in AC2. The resulting quotient is placed in AC1 and the remainder is placed in AC0. Carry←-Carry. If overflow is detected, AC0,1, and 2 remain unchanged.

    P      
integer

The following instruction is used to convert a 32 bit to a 16 bit integer. Carry←--Carry and Overflow←--ALU output. The format of this instruction is:



**CVWN** Convert the contents of the AC into a 16 bit integer. This instruction sets OVERFLOW if the most significant 17 bits of the specified AC before conversion are not the same. The specified AC is loaded with a sign extended integer even though an overflow is detected.

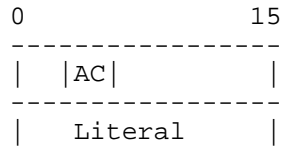
The following two instructions have a 16 bit immediate that is sign extended to 32 bits before being used to effect a fixed point accumulator.

**NLDAI** - Narrow Load Immediate

NLDAI has the following format:

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



The 16 bit two's complement literal is sign extended to 32 bits and loaded into the specified accumulator. Carry←Carry and Overflow←0.

WNADI - Wide Add with Narrow Immediate

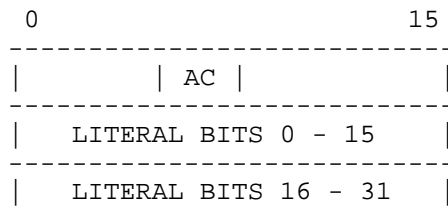
WNADI has the following format:



The 16 bit two's complement literal is sign extended to 32 bits and algebraically added to the specified accumulator. Carry ← ALU output. Overflow is set to the ALU output.

The following 32 bit fixed point arithmetic instructions effect a fixed point accumulator and have a 32 bit in-line literal or 2 bit literal:

The following two instruction have a 32 bit immediate and have the following format:



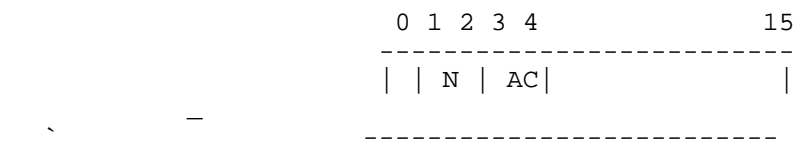
Data General Corporation  
Company Confidential

Rev. 7

WLDAI Load a 32 bit literal into AC. Carry<-Carry and Overflow<-0.

WADDI Add a 32 bit literal to AC. Carry<-Alu Output and OVERFLOW is set to the ALU output.

The format of the following 2 instructions are:



WADI Add N+1 to ACD. Carry<-Alu Output and OVERFLOW is set to the ALU output.

WSBI Subtract N+1 from ACD. Carry<-Alu Output and OVERFLOW is set to the ALU output.

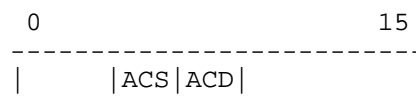
Data General Corporation  
Company Confidential

Rev. 7



## 7.6 Fixed Point Double Word Logicals

The EAGLE instruction set provides 32 bit logical operations on the 32 bit fixed point accumulators. The following instructions specify two accumulators: source (ACS) and destination (ACD) with ACD loaded with the results of ACD op ACS. These instructions have the format:



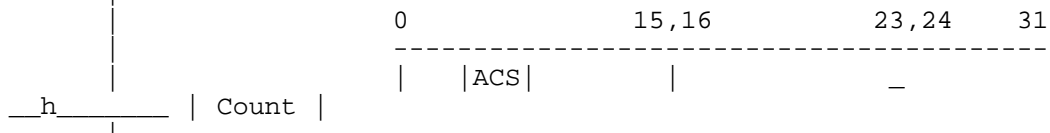
- ```
*  WAND      32 bit and.
*  WCOM      32 bit complement.
*  WIOR      32 bit inclusive or.
*  WXOR      32 bit exclusive or.
*  WANC      32 bit and with complement ACS.
*  WLSH      32 bit logical shift.
```

NOTE :

WLSH uses bit positions 24-31 of ACS for the shift count and shift direction. The Eclipse LEF can be used to load this shift specifier.

WLSHI - Wide Logical Shift Immediate.

The format of this instruction is:



Logically shifts the contents of ACD either left or right depending on the two's complement number contained in bits 24-31 of the immediate. If the number is positive, shifting is to the left and zeros fill vacated bit positions. If the number is negative, shifting is to the right and zeros fill vacated bit positions. Carry←Carry and

Data General Corporation  
Company Confidential

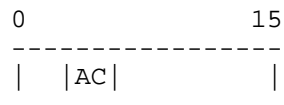
Rev. 7

|

Overflow<-0.

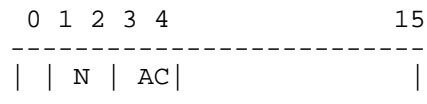
WMOVR - Move AC right one bit position.

The format of this instruction is:



The contents of the specified AC are logically shifted right one bit position, zero filled. Carry<-Carry and Overflow<-0.

The format of the following instruction is:



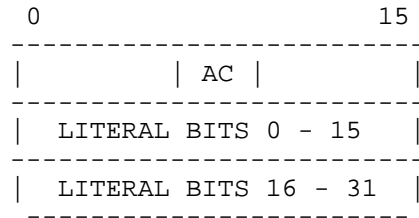
WLSI - Wide Logical Shift Immediate.

The specified accumulator is logically shifted left the number of positions as specified by the immediate field N, plus 1. Carry <-Carry and Overflow <- 0.

\_\_\_\_\_p\_\_\_\_\_ The following 32 bit fixed point logical instructions effect a fixed point accumulator and have a 32 bit in-line literal in the format:

Data General Corporation  
Company Confidential

Rev. 7

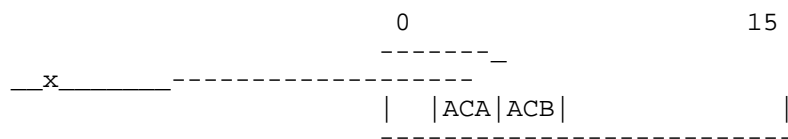


- \* WANDI    And with a 32 bit literal.
- \* WIORI    Inclusive or with a 32 bit literal.
- \* WXORI    Exclusive or with a 32 bit literal.

Data General Corporation  
Company Confidential

## 7.7 Double Word Compare ACs

The EAGLE instruction set provides fixed point accumulator comparison instructions which increment the PC if the condition is true. If the accumulator specifiers are equal, then the AC is compared to zero. They have the format:  
 Carry<-Carry and OVERFLOW<-0



- \* WUSGE Unsigned skip if ACA >= ACB.
- \* WUSGT Unsigned skip if ACA > ACB.
- \* WSEQ Skip if ACA = ACB.
- \* WSNE Skip if ACA <> ACB.
- \* WSGE Signed skip if ACA >= ACB.
- \* WSLE Signed skip if ACA <= ACB.
- \* WSGT Signed skip if ACA > ACB.
- \* WSLT Signed skip if ACA < ACB.

Data General Corporation  
Company Confidential

Rev. 7



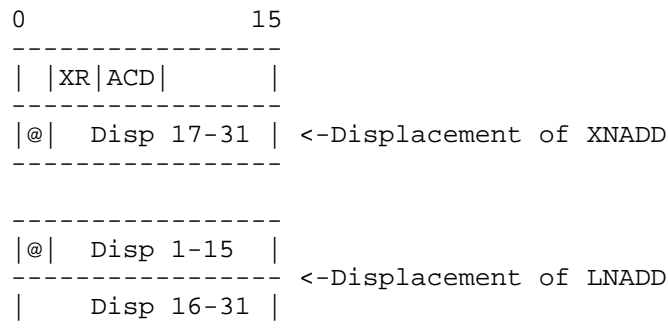
### 7.8 Fixed Point Word Indexed Address Arithmetics

The following 8 instructions perform operations between the word memory operand and the least significant 16 bits of the specified fixed point accumulator. The most significant 16 bits of ACD are ignored. The operation performed is:  $ACD \langle SEX \text{ to } 32 \rangle \leftarrow ACD \langle 16:31 \rangle .OP. \langle \text{memory word} \rangle$ . Upon completion of the operation, the 16 bit result is sign extended to 32 bits prior to the load of the 32 bit ACD. The contents of the referenced memory location remain unchanged. The ALU carry out and overflow refer to manipulations on two 16 bit fixed point operands. If a fixed point overflow is detected and OVK is a 1 a fixed point fault occurs. After the Wide Return Block is pushed, AC0 is loaded with the address of the instruction that caused the fixed point fault. Please see the chapter on Fault Mechanism for a complete description.

The new instructions are:

L/X NADD - Fixed Point Narrow Add with memory word

The format of these instructions are:



The 16 bit fixed point integer referenced in memory is algebraically added to the 16 least significant bits of ACD. ACD is loaded with the sign extended 16 bit sum (sign extended to 32 bits). The referenced memory location remains unchanged. Carry<-ALU and Overflow<-ALU

L/X NSUB - Fixed Point Narrow Subtract with memory word

Data General Corporation  
Company Confidential

Rev. 7



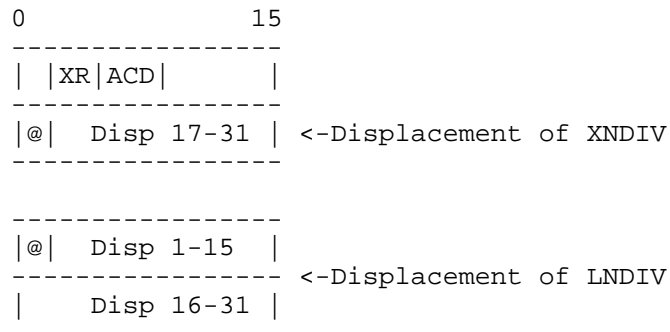
Data General Corporation  
Company Confidential

Rev. 7

extended 16 bit product(sign extended to 32 bits). The referenced memory location remains unchanged. Carry<-Carry. Overflow <- is set to 1 if the product requires more than 16\_ bits of precision. Otherwise, overflow is reset to 0.

#### L/X NDIV - Fixed Point Narrow Divide with memory word

The format of these instructions are:



The least significant 16 bits of ACD(dividend) is divided by the word referenced in memory (divisor) and the 16 bit quotient (sign extended to 32 bits) is loaded into ACD. Carry<-Carry. Overflow is set to 1 if the divisor is zero or the most negative integer is divided by -1. Otherwise, Overflow is 0. If overflow is detected, ACD remains unchanged. The referenced memory location remains unchanged.

Data General Corporation  
Company Confidential

Rev. 7

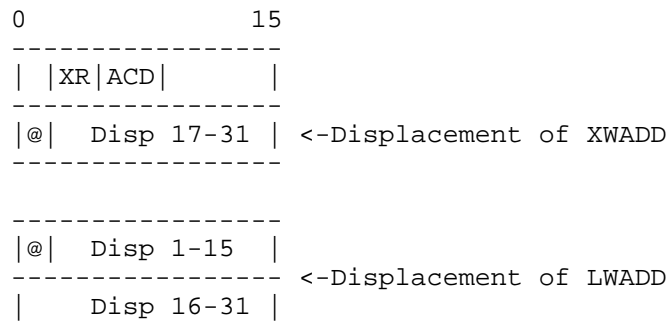
## 7.9 Fixed Point Double Word Indexed Address Arithmetics

The following 8 instructions perform operations between the double word memory operand referenced and the specified fixed point accumulator. The operation performed is:  $ACD \leftarrow ACD.OP.<memory\_double\_word>$ . Upon completion of the operation, the 32 bit result is loaded into the 32 bit ACD. The contents of the referenced memory location remain unchanged. The ALU carry out and overflow refer to manipulations on two 32 bit fixed point operands. If a fixed point overflow is detected and OVK is a 1 a fixed point fault occurs. After the Wide Return Block is pushed, AC0 is loaded with the address of the instruction that caused the fixed point fault. Please see the chapter on Fault Mechanism for a complete description.

The new instructions are:

L/X WADD - Fixed Point Wide Add with memory double word

The format of these instructions are:



The 32 bit fixed point integer referenced in memory is algebraically added to the contents of ACD. ACD is loaded with the 32 bit sum. The referenced memory location remains unchanged. Carry<-ALU and Overflow<-ALU

L/X WSUB - Fixed Point Wide Subtract with memory doubleword

The format of these instructions are:

Data General Corporation  
Company Confidential

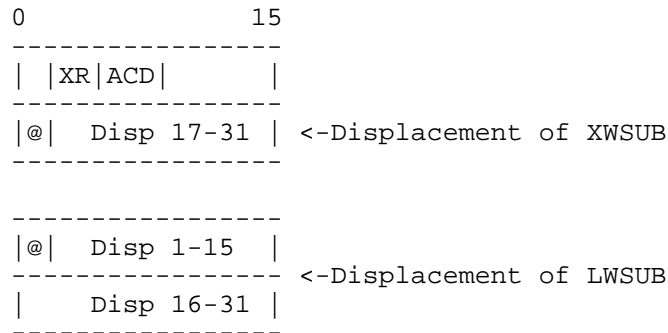
Rev. 7



## 7.9 Fixed P\_

\_\_\_\_\_oint Double Word Indexed Address Arithmetics

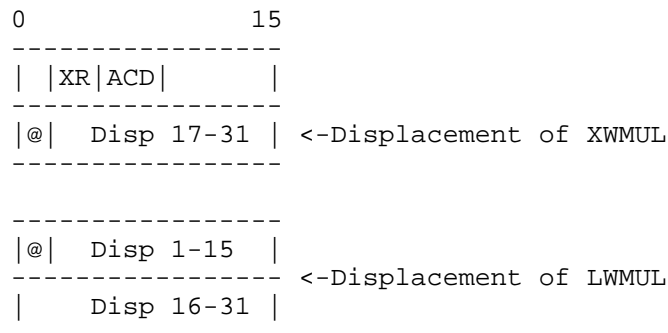
7-31



The 32 bit fixed point integer referenced in memory is algebraically subtracted from the contents of of ACD. ACD is loaded with 32 bit difference. The referenced memory location remains unchanged. Carry<-ALU and Overflow<-ALU

L/X WMUL - Fixed Point Wide Multiply with memory double word

The format of these instructions are:



The 32 bit fixed point integer referenced in memory is algebraically multiplied by the contents of ACD. ACD is loaded with the least significant 32 bits of the 64 bit intermediate product. The referenced memory location remains unchanged. Carry<-Carry. Overflow<- is set to 1 if the product requires more than 32 bits of precision. Otherwise, Overflow is reset to 0.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

The format of these instructions are:

|       |            |                         |
|-------|------------|-------------------------|
| 0     | 15         |                         |
| ----- |            |                         |
|       | XR ACD     |                         |
| ----- |            |                         |
| @     | Disp 16-31 | <-Displacement of XWDIV |
| ----- |            |                         |
|       |            |                         |
| ----- |            |                         |
| @     | Disp 1-15  |                         |
| ----- |            | <-Displacement of LWDIV |
|       | Disp 16-31 |                         |

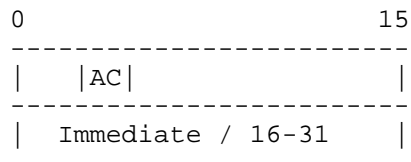
The contents of ACD(dividend) is divided by the double word referenced in memory (divisor) and the 32 bit quotient is loaded into ACD. Carry<-Carry. Overflow is set to 1 if the divisor is zero or the most negative integer is divided by -1. Otherwise, Overflow is reset to 0. If overflow is detected, ACD remains unchanged. The referenced memory location remains unchanged.

Data General Corporation  
Company Confidential

## 7.10 Skip Immediates

The following instruction perform\_0 a comparison between a fixed point AC and an in-line immediate. Carry and Overflow are unchanged. If the comparison is true the next 16-bit word is skipped. If the comparison is false the next 16-bit word is executed.

The following four instructions have the following format:



WSEQI - Skip if Equal Immediate

The 16-bit immediate is sign extended to 32-bits. This sign extended value is compared to the contents of the specified fixed point accumulator. If the AC is equal to the sign extended value, skip the next word, otherwise continue.

WSGTI - Skip if Greater Than Immediate

The 16-bit immediate is sign extended to 32-bits. This sign extended value is compared to the contents of the specified fixed point accumulator. If the AC is greater than the sign extended value, skip the next word, otherwise continue.

WSLEI - Skip if Less Than or Equal Immediate

The 16-bit immediate is sign extended to 32-bits. This sign extended value is compared to the contents of the specified fixed point accumulator. If the AC is less than or equal to the sign extended value, skip the next word, otherwise continue.

WSNEI - Skip if Not Equal Immediate

\_\_8\_\_\_\_\_/80

Data General Corporation  
Company Confidential

Rev. 7

|

The 16-bit immediate is sign extended to 32-bits. This sign extended value is compared to the contents of the specified fixed point accumulator. If the AC is not equal to the sign extended value, skip the next word, otherwise continue.

The following two instructions perform an unsigned comparison with a 32-bit immediate. The format of these instructions are:

|       |                 |
|-------|-----------------|
| 0     | 15              |
| ----- |                 |
|       | AC              |
| ----- |                 |
|       | Immediate 0-15  |
| ----- |                 |
|       | Immediate 16-31 |
| ----- |                 |

WUGTI - Skip if unsigned greater than immediate

An unsigned comparison is performed between the 32-bit immediate and the contents of the fixed point AC. If the AC is greater than the immediate, skip the next word, otherwise continue.

WULEI - Skip if unsigned less than or equal immediate

An unsigned comparison is performed between the 32-bit immediate and the contents of the fixed point AC. If the AC is less than or equal to the immediate, skip the next word, otherwise continue.

### 7.11 Memory Add/Sub Short Immediate

The following 8 fixed point instructions add or subtract a short immediate (1,2,3, or 4) from a 16 or 32-bit memory location.

\_\_\_\_\_  
 @ \_\_\_\_\_ | <mem><=<mem>.OP.<N+1>. No accumulator modification occurs.  
 Carry

and Overflow are set as a result of the arithmetic operation between the specified memory location and the short immediate. These instructions are NOT indivisible. If a fixed point trap occurs, the contents of the specified memory location are

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

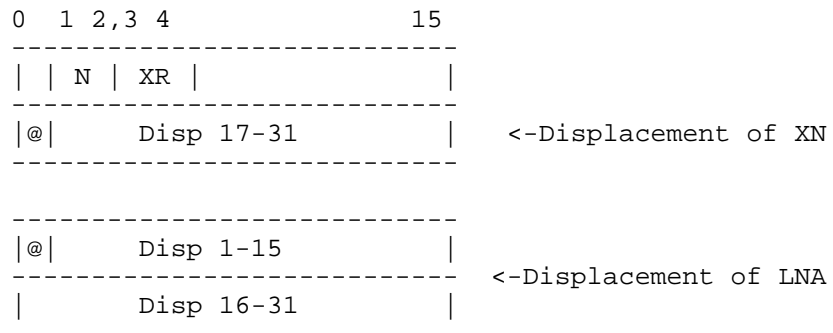
|



unspecified. These instructions are:

L/X NADI - Fixed Point Narrow Add to memory with immediate:

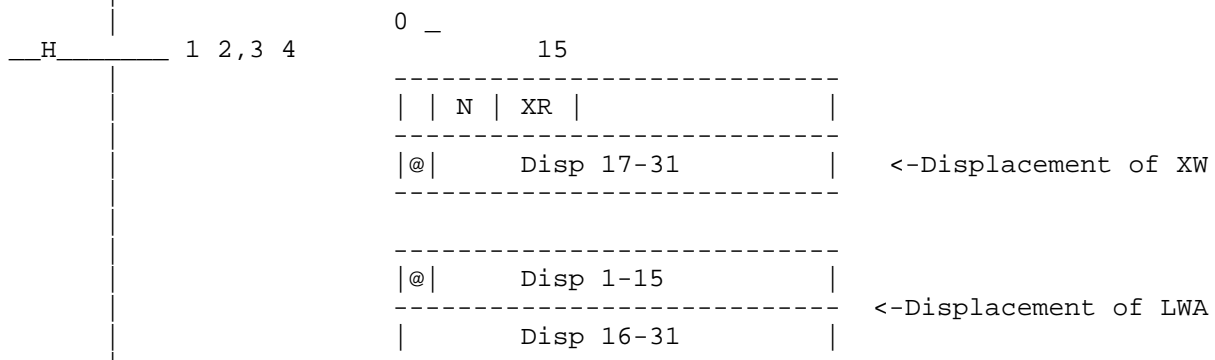
The format of these instructions are:



The value of N+1 (1,2,3,4) is algebraically added to the 16-bit fixed point integer referenced in memory. Carry, Overflow <-ALU output.

L/X WADI - Fixed Point Wide Add to memory with immediate:

The format of these instructions are:



Data General Corporation  
Company Confidential

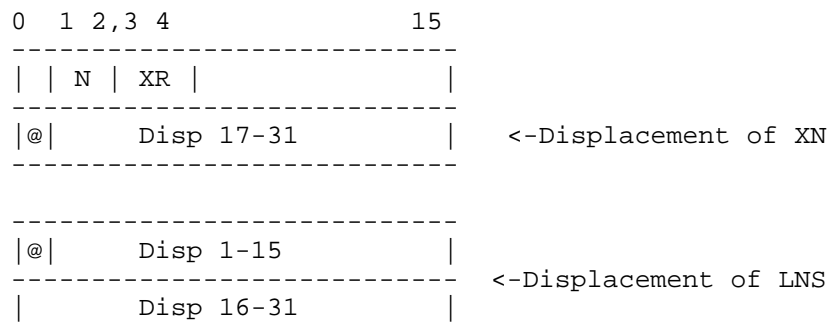
Rev. 7

|

The value of N+1 (1,2,3,4) is algebraically added to the 32-bit fixed point integer referenced in memory. Carry, Overflow <-ALU output.

L/X NSBI - Fixed Point Narrow Subtract from memory with immediate:

The format of these instructions are:



$\overline{P}$

The value of N+1 (1,2,3,4) is algebraically subtracted from the 16-bit fixed point integer referenced in memory. Carry, Overflow <-ALU output.

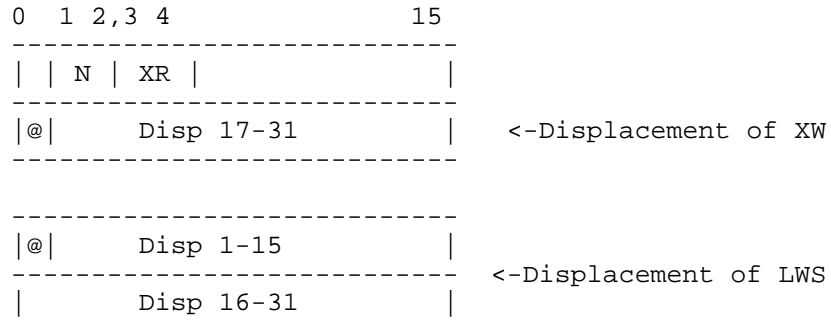
L/X WSBI - Fixed Point Wide Subtract from memory with immediate:

The format of these instructions are:

Data General Corporation  
Company Confidential

Rev. 7

|



The value of N+1 (1,2,3,4) is algebraically added to the 16-bit fixed point integer referenced in memory. Carry, Overflow <-ALU output.

#### 7.12 DO LOOP

The following instructions are used to perform loop control when the the loop variable is incremented by 1. These instructions increment a memory location by 1. Setting Carry and Overflow with the output of the ALU after the increment is performed, and then comparing the incremented\_

X\_d value to the 32-bit fixed point AC. The compare is always between 32-bits. 16-bit loop operands are signed extended to 32-bits prior to the compare. If the incremented memory operand is less than or equal to the AC, a 16-bit PC relative termination offset is skipped. If the incremented memory operand is greater than the AC, the termination offset is added to the 1 + the address of the word containing the DO LOOP instruction opcode. Additionally the incremented memory operand is moved to the AC containing the termination comparand. If a fixed point overflow trap occurs (on the increment), the contents of the specified memory location and the PC value in the return block are undefined. AC0 in the fixed point trap handler properly references the address of the DO LOOP instruction.

The format of these instructions are:

Data General Corporation  
Company Confidential

Rev. 7

|

|                    |                              |
|--------------------|------------------------------|
| 0                  | 15                           |
| AC XR              |                              |
| @  Disp 17-31      | <-X Prefix                   |
| Termination offset | <-Added to PC+1 if loop end. |
| Normal Exist       | <-Iterate again              |

|                    |                              |
|--------------------|------------------------------|
| 0                  | 15                           |
| AC XR              |                              |
| @  Disp 1-15       | <- L Prefix                  |
| Disp 16-31         |                              |
| Termination offset | <-Added to PC+1 if loop end. |
| Normal Exist       | <-Iterate again              |

L/X NDO - Do until 16-bit memory operand is greater than AC.

L/X WDO - Do until 32-bit memory operand is greater than AC.

Data General Corporation  
Company Confidential

Rev. 7



## 7.13 Program Flow

When the PC is altered by any C350 program flow instructions (those which use a 15 bit PC), bits 4 - 16 of the PC become all "0". EAGLE instructions have been provided to allow modification of the 31 bit program counter (PC). Although some of the program flow instructions have 31 bit displacements, in most cases only 28 bits are significant since only WPOPB, WRSTR, XCALL, LCALL and WRTN can legally effect the current ring number.

For all other program flow instructions, only the least significant 28 bits of the PC are altered. The ring field of the effective address is ignored. Thus all PC relative jumps wraparound within the current segment. If the effective address is a pointer obtained via indirection, the least significant 28 bits of the pointer replace the least significant 28 bits of the PC. All instructions in th\_

h\_\_\_\_\_is section follow these rules. LDSP is somewhat different in that the pointer is interpreted as a self-relative jump.

For all Eclipse PC save type instructions (e.g., JSR, EJSR, PSHJ, etc. ), the operation PC+1 or PC+2, is modulo  $2^{*}15$ .

For all the following instructions, Carry<-Carry and OVERFLOW<-0.

The instructions with a 31 bit displacement have the following format:

| 0     | 15                |
|-------|-------------------|
| ----- |                   |
|       | INDEX             |
| ----- |                   |
| @     | DISP BITS 1 - 15  |
| ----- |                   |
|       | DISP BITS 16 - 31 |
| ----- |                   |

LJMP Load PC with 31 bits of the effective address.

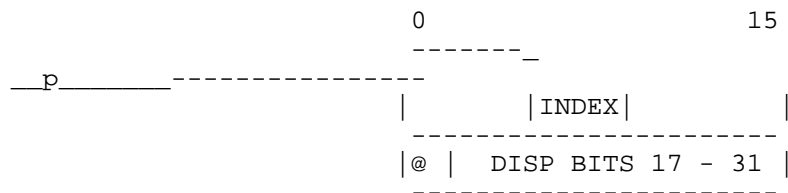
LJSR Load AC3 with 31 bit PC+3 and load the PC with 31 bits of the effective address. The operation PC+3 is modulo  $2^{*}28$ . Thus the return address loaded into AC3 always references the current ring.

Data General Corporation  
Company Confidential

Rev. 7

LPSHJ Push 31 bit PC+3 onto the EAGLE stack and load PC with 31 bits of the effective address. The operation PC+3 is modulo  $2^{**}28$ . Thus the return address loaded onto the stack always references the current ring.

Those with a 15 bit displacement have the following format:



XJMP Load PC with a 31 bit effective address.

XJSR Load AC3 with 31 bit PC+2 and load PC with a 31 bit effective word address. The operation PC+2 is modulo  $2^{**}28$ . Thus the return address loaded into AC3 always references the current ring.

XPSHJ Push 31 bit PC+2 onto the EAGLE stack, and set PC to 31 bits of effective address. The operation PC+2 is modulo  $2^{**}28$ . Thus the return address loaded onto the stack always references the current ring.

C/350 instructions of the class that loads AC3 with the address of the next instruction (JSR), or push the address of the next instruction onto the C/350 stack (PUSHJ), perform the operation PC+1 modulo  $2^{**}15$ .

The EAGLE architecture includes a PC relative jump instruction. Indirect addressing is not specifiable. This instruction is one word long. This is especially useful with C350 skip instructions since the C350 JMP instruction cannot be used for addressing above 64KB. An 8 bit displacement is obtained from bits 1-4 and 6-9 of the C350 XOP1 instruction. This displacement is sign extended to 31 bits and added to the PC. The format of this instruction is:

For all the following instructions, Carry<-Carry and OVERFLOW<-0.

Rev. 7

Data General Corporation  
Company Conf\_

  x  identical

|       |        |   |   |        |      |        |
|-------|--------|---|---|--------|------|--------|
| 0     | 1      | 4 | 5 | 6      | 9,10 | 15     |
| ----- |        |   |   |        |      |        |
| 1     | DISP-4 |   | 0 | DISP-4 |      | 111000 |
| ----- |        |   |   |        |      |        |

\* WBR      Load PC with 31 bit PC + disp.

The following instruction requires no index register or displacement:

\* WPOPJ    POP 31 bit address from stack into PC.

The following instruction deals with a 31 bit displacement and expects a table of 28 bit self relative PC addresses.

LDSP - Long Dispatch

The dispatch instruction has the following format:

|       |                   |    |  |  |                       |  |
|-------|-------------------|----|--|--|-----------------------|--|
| 0     |                   |    |  |  | 15                    |  |
| ----- |                   |    |  |  |                       |  |
|       | INDEX             | AC |  |  |                       |  |
| ----- |                   |    |  |  |                       |  |
| @     | DISP BITS 1 - 15  |    |  |  | <- This address, E,   |  |
| ----- |                   |    |  |  |                       |  |
|       | DISP BITS 16 - 31 |    |  |  | references a Dispatch |  |
| ----- |                   |    |  |  |                       |  |
|       |                   |    |  |  | Table of self         |  |
|       |                   |    |  |  | relative addresses.   |  |

Dispatch through a table of 28 bit self relative addresses indexed by the 31 bit AC. Similar to DSPA except all addresses are 31 bits. The ring field of the fetched table entry is ignored. The 28 bit self relative address in the table entry is added to the address of the table entry. Wraparound occurs within this 28 bit offset. A ring crossing can not occur. The effective address, E, references a table of self relative addresses in the current segment. Thus, bits 1-3 of E, and bits 1-3 of any levels of indirection, are always interpreted as the current segment.

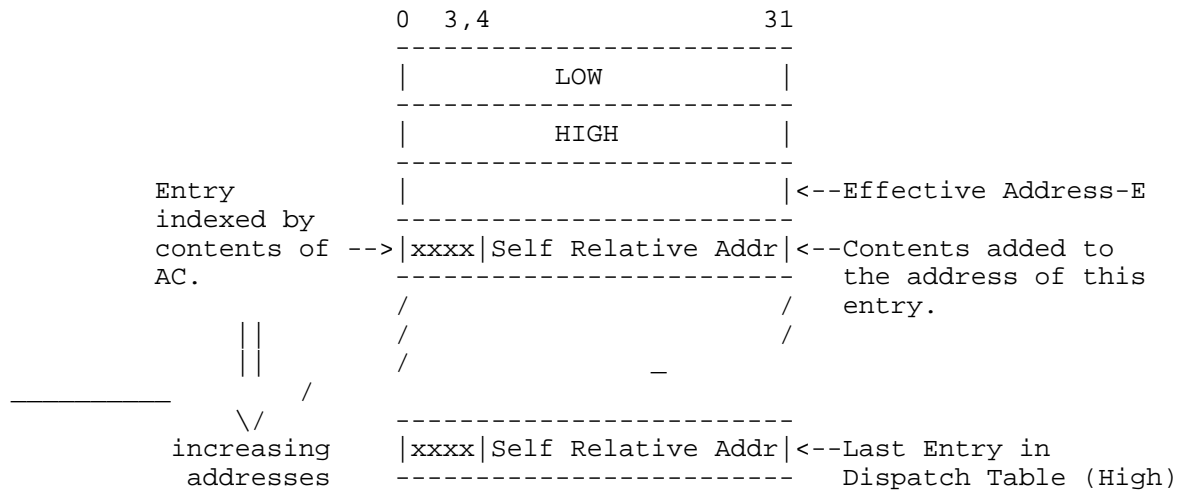
---

The two's complement number contained in the accumulator is compared to the two limit words. If the number in the accumulator is less than LOW or greater than HIGH, sequential operation continues with the instruction immediately after the LDSP instruction. If the number is greater than

Data General Corporation  
Company Confidential

Rev. 7

or equal to LOW and less than or equal to HIGH, the instruction fetches the double word at location  $E-2*(LOW-number)$ . This double word is interpreted as a self-relative address in the current segment if all 32 bits are not all 1. If all 32 bits are all 1, sequential operation continues with the instruction immediately after the LDSP instruction. The structure of the dispatch table is:



Data General Corporation  
Company Confidential

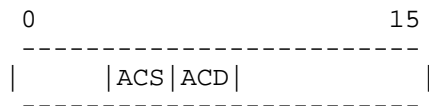
Rev. 7



## 7.14 Double Word Bit Operations

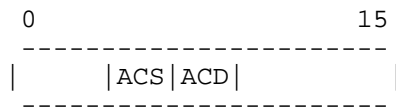
The following bit instructions expect a bit pointer consisting of a word pointer in ACS and a bit offset in ACD. If ACS and ACD specify the same accumulator, then the word pointer is assumed to be 0 in the current ring, and the AC contains a 32 bit bit pointer.

For the following instructions Carry is unchanged and OVERFLOW is 0.



- \* WBTO     Set bit to one.
- \* WBTZ     Set bit to zero.
- \* WSZB     Skip on zero bit.
- \* WSNB     Skip on non-zero bit.
- \* WSZBO    Skip on zero bit and set bit to one.  
The bit is tested and set to 1 in one memory cycle.

The following bit instructions operate on the contents of ACS and place the result in ACD. The format is:



- \* WLOB     Locate leading bit
- \* WLRB     Locate and reset leading bit

- 
- \* WCOB     Count bits

Data General Corporation  
Company Confidential

Rev. 7

## 7.15 Character Instructions

The following character instructions are identical to the corresponding C350 instructions except that word addresses in the ACs are 31 bits and byte addresses are 32 bits.

For the following Eagle instructions, when the move or scan is backwards, a check is performed to determine if a segment boundary would be crossed. If it is, the instruction is not executed and a protection is signalled (code=4 in AC1). The affected instructions are: WCMT, WCMV, and WCMP.

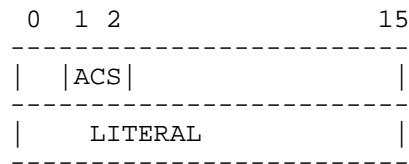
|      |                                                                                                                                                                                                            |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WLDB | Load byte (LDB) with 32 bit ACs.                                                                                                                                                                           |
| WSTB | Store byte (STB) with 32 bit ACs.                                                                                                                                                                          |
| WCTR | Character translate (CTR) with 32 bit ACs. Note: If the initial pointers reference inward a protection violation will occur.<br>)                                                                          |
| WCMT | Character move until true (CMT) with 32 ACs. Note: If the initial pointers reference inward a protection violation will occur.                                                                             |
| WCMV | Character move with 32 bit ACs. Note: If the initial pointers reference inward a protection violation will occur.                                                                                          |
| WCMP | Character compare with 32 bit ACs. Note: If the initial pointers reference inward a protection violation will occur.                                                                                       |
| WSCT | Character Scan until true. Same as WCMT except no data is stored in memory and AC2 is unused by the instruction.<br>Note: If t_<br>he initial pointers reference inward a protection violation will occur. |

Data General Corporation  
Company Confidential

Rev. 7

## 7.16 Skip on Word Masked Field

The following instructions provide a means to test if any or all selected bits in the least significant 16 bits of the specified accumulator are set to one. Carry <- Carry and OVERFLOW <- 0.



NSANA - Narrow Skip on Any Bit Set in Accumulator.

The literal field is ANDed with the least significant 16 bits of the specified accumulator. If any bits of this ANDed result is a 1, the next word is skipped. Otherwise the next sequential location is executed. The specified accumulator remains unchanged.

NSALA - Narrow Skip on All Bits Set in Accumulator.

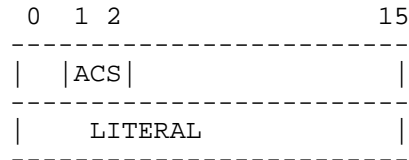
The literal field is ANDed with the complement of the least significant 16 bits of the specified accumulator. If all bits of this result are 0, the next word is skipped. Otherwise, the next sequential location is executed. The specified accumulator remains unchanged.

The following instructions provide means to test if any or all selected bits in the specified memory location\_

\_\_\_\_\_ are set to 1.

Data General Corporation  
Company Confidential

Rev. 7



NSANM - Narrow Skip on Any Bit Set in Memory.

The literal field is ANDed with the contents of the word in memory specified by the effective address contained in ACS. If any bit of this ANDed result is a 1, the next word is skipped. Otherwise, the next sequential location is executed. The specified memory location remains unchanged.

NSALM - Narrow Skip on All Bits Set in Memory.

The literal field is ANDed with the complement of the word in memory specified by the effective address contained in ACS. If all bits of this result are a 0, the next word is skipped. Otherwise, the next sequential location is executed. The specified memory location remains unchanged.

## 7.17 Skip on Double Word Masked Field

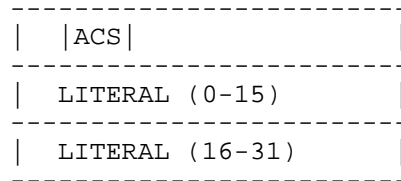
The following instructions provide a means to test if any or all selected bits in the specified double word are set to a 1. The format of these instructions are:

— ( —————

Rev. 7

Data General Corporation  
Company Confidential





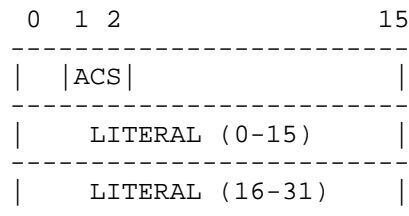
WSANA - Wide Skip on Any Bit Set.

The literal field is ANDed with the specified accumulator. If any bit of this ANDed result is a 1, the next word is skipped. Otherwise the next sequential word location is executed. The specified accumulator remains unchanged.

WSALA - Wide Skip on All Bits Set.

The literal field is ANDed with the complement of the specified accumulator. If all bits of this result are 0, the next word is skipped. Otherwise the next sequential location is executed. The specified accumulator remains unchanged.

The following instructions provide means to test if any or all selected bits in the specified double word memory location are set to 1.



WSANM - Skip on Any Bit Set in Double Word Memory.

The literal field is ANDed with the contents of the double word in memory specified by the effective address contained in ACS. If any bit of this ANDed result is a 1, the next word is skipped. Otherwise, the next sequential location is executed. The specified memory location remains unchanged.

Data General Corporation  
Company Confidential

Rev. 7

WSALM - Skip on All Bits Set in Double Word Memory.

The literal field is Anded with the complement of the double word in memory specified by the effective address contained in ACS. If all bits of this result are a 0, the next word is skipped. Otherwise, the next sequential location is executed. The specified memory location remains unchanged.

The following two instructions test one bit in AC0. The format of these instructions are: Carry <- Carry and Overflow <- 0

|       |       |    |       |
|-------|-------|----|-------|
| 0,1   | 3 6   | 10 | 11 15 |
| ----- |       |    |       |
|       | BIT # |    | BIT#  |
| ----- |       |    |       |

WSKBO - Skip on Bit One.

The specified bit in AC0 is tested. If this bit is a 1, the next sequential word is skipped. Otherwise the next word is executed.

WSKBZ - Skip on Bit Zero.

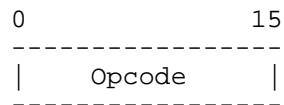
The specified bit in AC0 is tested. If this bit is a 0, the next sequential word is skipped. Otherwise the next word is executed.

Data General Corporation  
Company Confidential

Rev. 7

## 7.18 Processor Status Register and Carry Instructions

The following three instructions: load, store, and skip on the flags necessary to control fixed point overflow. The load and store specify an accumulator. The format of these instructions are:



## LPSR - Load Processor Status Register

The OVK, OVR, and IRES flags are loaded left justified, zero filled into AC0.

## SPSR - Store Processor Status Register

Bits 0, 1 and 2 of AC0 are loaded into the OVK, OVR and IRES flags respectively.

## SNOVR - Skip on OVR Reset

The OVR flag is tested. If it is a 0, the next sequential word is skipped.

Data General Corporation  
Company Confidential

The following instructions facilitate efficient setting and resetting of internal processor flags. The format of these instructions are:



CRYTO - Carry To One

The Carry is unconditionally set to a 1. Overflow <-- 0.

CRYTZ - Carry To Zero

The Carry is unconditionally reset to a 0. Overflow <-- 0.

CRYTC - Carry To Complement

The Carry is complemented. Overflow <-- 0.

FXTD - Fixed Point Trap Disable

The OVK flag is unconditionally reset to a 0, thereby disabling the fixed point overflow traps. Carry <-- Carry

FXTE - Fixed Point Trap Enable

The OVK flag is unconditionally set to a 1. The OVR flag is unconditionally reset to 0. Carry <-- Carry. This instruction provides an efficient means to enable fixed point overflow traps.

Data General Corporation  
Company Confidential

Rev. 7



## 7.19 Others

## WBLM - Wide Block Move

AC1 specifies the length and direction of the move. If AC1 is positive, movement is from the lowest memory location to the highest (ascending). AC1 contains the unsigned number of words to be moved. If AC1 is negative, movement is from the highest memory location to the lowest (descending). AC1 contains the two's complement of the number of words to be moved. The source address of the words to be moved is specified by the word pointer contained in AC2. The destination of the words to be moved is specified by the word pointer contained in AC3. Upon completion o\_

\_\_\_\_\_H\_\_\_\_\_f the

instruction, AC1 contains zeroes, and AC2 and AC3 point to the word following the last word in their respective fields. If movement is backwards and a ring crossing would occur, the instruction is not executed and a protection fault is signalled (code=4 in AC1)

WCLM - Compare to limits using 32 bit AC's.  
The format of this instruction is:

```

      0                      15
      -----
      | |ACS|ACS|          |
      -----

```

Compares a signed integer with two other integers and skips if the first integer is between the other two. The 32 bit accumulators determine the location of the three integers.

Compares the signed, two's complement integer in ACS to two signed, two's complement limit values, L and H. If the number in ACS is greater than or equal to L and less than or equal to H, the next sequential word is skipped. If the number in ACS is less than L or greater than H, the next sequential word is executed.

If ACS and ACD are specified as different accumulators, the address of the limit value L is contained in ACD. The limit value H is contained in the double word following L. Bit 0 of ACD is ignored.

If ACS and ACD are specified as the same accumulator, then the integer to be compared must be in that AC and the limit

Data General Corporation  
Company Confidential

Rev. 7

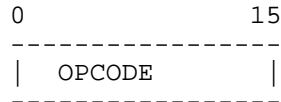
P

values L and H must be the two double words following the instruction. L is the first double word and H is the second double word. The next sequential word is the fifth word following the instruction. Carry<-Carry and Overflow<-0

Data General Corporation  
Company Confidential

## BKPT - Break Point

The format of this instruction is:



A wide return block is pushed onto the present Eagle stack. The value of the PC in this block points to this instruction. A check for stack overflow is performed upon completion of the wide return block push. If a stack overflow is detected, a stack fault is signalled (code=0 in AC1). A wide jump indirect thru locations 10-11 of the current segment is then performed. PSR (OVR, OVR, and IRES) is reset to 0 after the return block is pushed. Carry is unchanged.

## PBX - Pop Block and Execute

The format of this instruction is:



This instruction is used to return from breakpoints. The PBX instruction performs the following:

- 1) The lower order 16 bits (bits 16-31) of AC0 are temporarily saved in internal processor state.
- 2) A WPOPB instruction return block is popped. All the functionality described for the WPOPB instruction is performed with the exception of the last step. That is instruction execution does not continue with the updated value of the PC. The PC is loaded however.

The popped PC must reference a BKPT instruction. If this

Data General Corporation  
Company Confidential

Rev. 7

popped PC does not reference a BKPT instruction indeterminate actions will occur. The WPOPB instruction should be used to return to deleted breakpoints.

- 3) The 16 bits saved in step 1) is executed.
- 4) The state of the processor flags after execution of the above step are a function of the instruction executed.
- 5) If the executed instruction does not directly alter the PC (e.g., LJMP) the next instruction executed is located at the memory location referenced by the popped PC value plus the length of the instruction executed on step 3).
- 6) In effect, the 16 bits saved in step 1) are "SUBSTITUTED" for the word referenced by the popped PC.

— , ———

Data General Corporation  
Company Confidential

Rev. 7



## WCIS - Wide Commercial Instruction

The format of this instruction is

|        |       |                  |
|--------|-------|------------------|
| 0      | 15,16 | 31               |
| -----  |       |                  |
| OPCODE |       | Additonal Instr. |
| -----  |       |                  |

The second word of the instruction is interpreted as an opcode. Presently, no instruction uses this format.

## WXOP - Wide Extended Operation.

The format of the WXOP instruction is:

|       |                |      |     |
|-------|----------------|------|-----|
| 0     | 1 1 1 1 1 2    | 2 2  | 3 3 |
|       | 5 6 7 8 9 0    | 3 4  | 0 1 |
| ----- |                |      |     |
| WXOP  | acs  acd  0000 | XOP# | 0   |
| ----- |                |      |     |

Pushes a return block onto the present Eagle stack. Places the stack address of ACS into AC2; (that is the address of the double word in the stack that the accumulator specified by ACS occupies) places the address in the stack of ACD into AC3. Memory locations 12-13 (octal) must contain the WXOP origin address, the starting address of a 128 (decimal) double word table of addresses. These addresses are the starting location of the various WXOP operations.

h \_\_\_\_\_ WXOP

Two time the operation number in the \_\_\_\_\_ instruction is added to the WXOP origin address to produce the address of a double word in the WXOP table. This double word is fetched and treated as an intermediate address in an effective address calculation. After the indirection chain, if any, has been followed, the instruction places the effective address in the program counter. The contents of AC0, AC1, and the WXOP origin address remain unchanged. All addresses are constrained to be in the current segment. The returned block is configured so that the WXOP procedure can return control to the calling program via the WPOPB instruction.

Carry<--Carry and Overflow<--0

Data General Corporation  
Company Confidential

Rev. 7

WWCS - Wide Enter WCS .

The format of the WWCS instruction is:

|       |            |   |
|-------|------------|---|
|       | 1 1        | 3 |
| 0     | 5 6        | 1 |
| ----- |            |   |
| WWCS  | Entry Info |   |
| ----- |            |   |

When the WCS is installed the microprogram entry point is specified in bits 16-31 of the entry information. The number of entry points and the instruction bits which specify these entry points are defined on a implementation specific basis. When WCS is installed the us\_

p er defined

microprogram determines the state of Carry and Overflow. When WCS is not installed, this instruction causes no operation to be performed.

LCS - Load Control Store.

If the control store exists in an implementation, if it is writable, and if an external means is not available for bootstrapping, then this instruction is used for loading microcode. This instruction is IO protected, and becomes a LEF if the LEF bit is enabled. The structure of the microcode loaded and the use of internal state to aid in the microcode load is implementation specific. The format of this instruction is:

|       |    |
|-------|----|
| 0     | 15 |
| ----- |    |
| LCS   |    |
| ----- |    |

WAP - Wide Array Processor Operation.

The format of this instruction is:

Data General Corporation  
Company Confidential

Rev. 7

|

| 0   | 15,16  | 31 |
|-----|--------|----|
| WAP | APOP # |    |

When the Eagle Array Processor option is installed, bits 16-31 are used to specify one of the AP opcodes. When the Eagle Array Processor option is not installed, this in-

x

struction causes no operation to be performed. When the Eagle Array Processor option is installed, the state of Carry and Overflow is determined by the APOP operation executed.

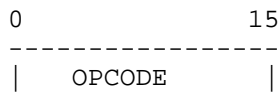
Data General Corporation  
Company Confidential

Rev. 7

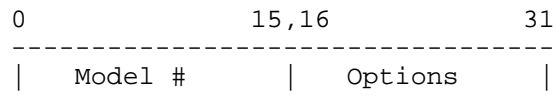
|

LCPID/ECLID - Load CPU identification.

The format of this instruction is:



The following 32 bit double word is loaded into AC0.



where model # is the binary value of the model number on the price list that marketing allocates to the processor. The options field indicates the various options that are installed on this machine. The interpretation of this field is model dependent. LCPID is an Eagle instruction. ECLID is an Eclipse instruction. When MMPUL is enabled, ECLID should be used. The user of ECLID when MMPUL is enabled must be aware of the following. AC0 is assumed to be 32 bits by this instruction. If an interrupt occurs, only the lea\_

```

    _st significant 16 bits of AC0 are saved.    Carry<--Carry
    and Overflow<--0

```

Data General Corporation  
Company Confidential

Rev. 7

|



NCLID - Nova version of LCPID.

NCLID places the CPU model number in AC0<16-31>, the microcode rev # in AC1<16-31>, and the memory size in AC2<16-31>. If AC1 returns 177777<octal>, then micorcode needs to be loaded (via LCS). Memory size is in blocks of 16 KW. This instruction is IO protected, and becomes LEF is the LEF bit is enabled. The format of this instruction is:

|       |    |
|-------|----|
| 0     | 15 |
| ----- |    |
|       |    |
|       |    |
| ----- |    |

DERR - Diagnostic Error.

The PC is pushed onto the wide stack, a 5-bit error zero extended to 32-bits is pushed onto the wide stack, and then a jump thru location 47 <octal> of the current segment is performed. Location 47 is indirectable. Carry and OVERFLOW are unchanged. The format of this instruction are:

|       |    |
|-------|----|
| 0     | 15 |
| ----- |    |
|       |    |
|       |    |
| ----- |    |

---

Rev. 7

Data General Corporation  
Company Confidential



Data General Corporation  
Company Confidential

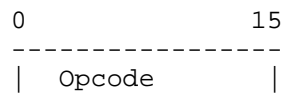
Rev. 7



AC0 contains a byte pointer. AC1 contains in bits 1-3 a ring number and 0's in bits 0, 4-31. The ring field of AC1 is compared to the ring field of the pointer in AC0 (bits 0-3). If the AC1 ring field is greater than the ring field of the AC0 pointer, the next word is executed. Otherwise, the next word is skipped. Carry<-Carry and Overflow<-0.

#### LPHY - Load Physical

The format of this instruction is:



- 1) If logical equals physical no operation is performed and the next word is executed. If the Eagle ATU is enabled continue.
- 2) AC1 contains a logical word address. The ring field of this address is compared with the current ring. If the ring field is greater than or equal to the current ring continue. If the ring field is less than the current ring, a protection fault ( AC1=4 ) occurs.
- 3) The SBR referenced by the contents of AC1 is examined. If the SBR is valid continue. If the SBR is invalid, execute the next word. The contents of AC0 are unchanged.
- 4) AC0 is loaded with the last resident PTE.
- 5) AC2 is loaded with the 32 bit (zero extended on the left) physical word address of the logical address contained in AC1, if no page or validity faults are indicated by the referenced PTE's. The next word is then skipped. If a page fault or validity fault is detected, AC2 is not loaded and the next word is then executed. The access bits in the referenced PTE's are not interpreted.

Data General Corporation  
Company Confidential

Rev. 7

6) Carry<-Carry and Overflow<-0.

--End of Chapter--

Data General Corporation  
Company Confidential



## Chapter 8

### Subroutine Call/Save/Return - Interrupt Return

The following instructions implement Call/Save/Return and Interrupt Return.

.XCALL/LCALL  
.WSAVR/WSAVS  
.WRTN  
.WPOPB  
.WRSTR  
.WSSVR/WSSVS

In a typical program the use of these instructions are as follows:

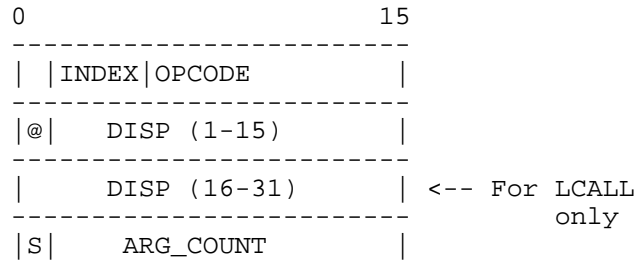
- 1) X or LCALL with WSAVR or WSAVS.
- 2) X or LJSR with WSSVR or WSSVS.
- 3) WRTN to return from a subroutine which built a return block using the WSSVR, WSSVS, WSAVR, or WSAVS instructions.
- 4) WPOPB to return to an interrupted program. The interrupt is either a I/O interrupt that occurred at intermediate level, a fault, or a deleted breakpoint.
- 5) WRSTR to return to a program which was interrupted by an I/O interrupt which occurred at base level.

XCALL/LCALL - Call Subroutine

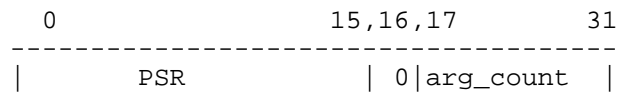
The L or XCALL instructions have the following format:

Data General Corporation  
Company Confidential

Rev. 7



The following figures are referenced in the description of these two instructions:



\_\_\_\_(\_\_\_\_\_

Figure 8-1. Call Created Double Word

If bit 0 of the arg\_count word is a 1, the top entry of the stack (referenced by the current value of ESP) is assumed to have the following structure.

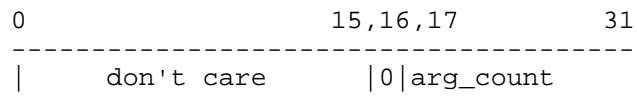


Figure 8-2. Top of Stack Entry

The effective address for inward ring calls is interpreted as:

Data General Corporation  
Company Confidential

Rev. 7

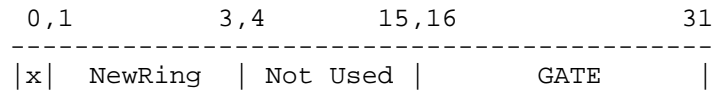


Figure 8-3. Inward Address

The following is a step by step description of the X and LCALL instructions.

1)The target address is evaluated. If an outward ring crossing is specified a protection fault (code=7 in AC1) is signalled. The return address in the pushed fault wide return block is undefined. AC0 properly contains the address of the CALL instruction. Otherwise continue.

2)Load AC3 with PC+3 for XCALL and PC+4 for LCALL. The operation PC+3 or PC+4 is modulo  $2^{**}28$ . Thus the return address loaded into AC3 always references the current ring. If an intra-ring call is indicated (same ring) proceed with steps 3 and 4. If an inward ring call proceed with step 5.

3)If the current ring is identical to the effective source, continue to steps 3A), 3B), and 4). If the current ring is not identical to the effective source, a check is made for a legal GATE (Fig. 3). If the specified GATE is not legal, a protection fault (code=6 in AC1) is signalled and the CALL is not performed. The return address in the pushed fault wide return block is undefined. AC0 properly contains the address of the CALL instruction. If the specified GATE is legal continue to steps 3A), 3B), and 4).

3A)If bit 0 of the arg\_count operand is a 0, a stack entry is created (Fig. 1) and pushed onto the present stack. If stack overflow occurs after the push a stack fault occurs and the subroutine call is not initiated. If bit 0 of the arg\_count operand is a 1, the top entry of the present stack is modified to contain the OVK and OVR flags.

3B)OVR is unconditionally reset to 0.

4)Load PC with the target address. Sequential execution

Data General Corporation  
Company Confidential

Rev. 7

continues with the newly updated value of the PC.

The following steps are for inward ring c\_

\_\_8\_\_\_\_alls.

5) If bit 0 of the arg\_count operand is a 0, a double word (Fig. 1) is created and temporarily saved in internal processor state. If bit 0 of the arg\_count operand is a 1, the present stack is popped (WSP<--WSP-2). The entry popped is modified so as to have the same structure as created above. This modified entry is temporarily saved in internal processor state.

6) A check is made for a legal GATE (Fig. 3). If the specified GATE is not legal, a protection fault (code=6 in AC1) is signalled and no ring crossing is performed. The return address in the pushed fault wide return block is undefined. AC0 properly contains the address of the CALL instruction. Otherwise continue.

7) The ring crossing is performed. WFP and WSP are stored into their respective location in page 0 of the current segment. A new stack is created for the Callee. WSP, WSL, and WSB are loaded from the appropriate page 0 locations of the Callee. WFP is unchanged. Ultimately, a WSAV(R or S) instruction loads WFP.

7A) Load the least significant 28 bits of the GATE into the least significant 28 bits of the PC. Load the new ring number into bits 1-3 of the PC. If a stack overflow is detected on step 8), the PC in the return block references the first instruction of the called subroutine.

8) A check is performed for stack overflow relative to the number of arguments to be transferred. If a stack overflow would occur, stack fault in this new ring. Otherwise continue.

9) Copy arguments from the Caller's stack onto the newly

\_\_@\_\_\_\_ created Callee's stack. (See the section entitled Gate Access/Ring Crossing in the Protection Chapter for a detailed description of this step)

10) Push the double word contained in internal processor state onto the Callee's stack. This is the double word created in step 5.

11) OVR is unconditionally reset to 0.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

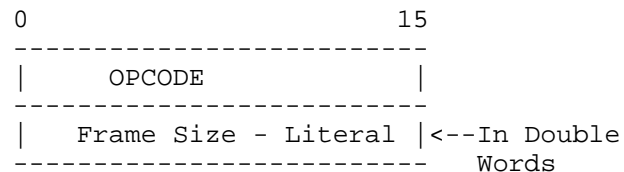


12)Continue sequential execution with the updated value of the PC.

If an EXECUTE protection violation occurs for the first instruction fetched, AC0 indicates an execute violation and the PC in the return block references the next sequential instruction.

#### WSAVS -Wide Save/ Set Overflow Mask

Saves the information required by the Wide Return instruction. This instruction is typically executed after a X or LCALL instruction. The format of this instruction is:



A wide return block is pushed onto the stack. After the fifth double word is pushed (the first double word of the six double word block w\_

H as pushed by the XCALL OR LCALL instruction ), the value of the stack pointer is placed in the frame pointer (WFP) and AC3.

The 16 bit unsigned integer ( the FRAME SIZE ) contained in the immediate field is multiplied by two and added to the stack pointer.

The Overflow Mask(OVK) is set to 1, thereby enabling integer overflow.

Before execution, the WSAVS instruction checks for stack overflow. If executing the instruction would result in a stack overflow, none of the above actions occur, and WSAVS transfers control to the stack fault routine. The program counter in the fault return block contains the address of the WSAVS instruction.

The structure of the return block pushed is as follows:

Data General Corporation  
Company Confidential

Rev. 7

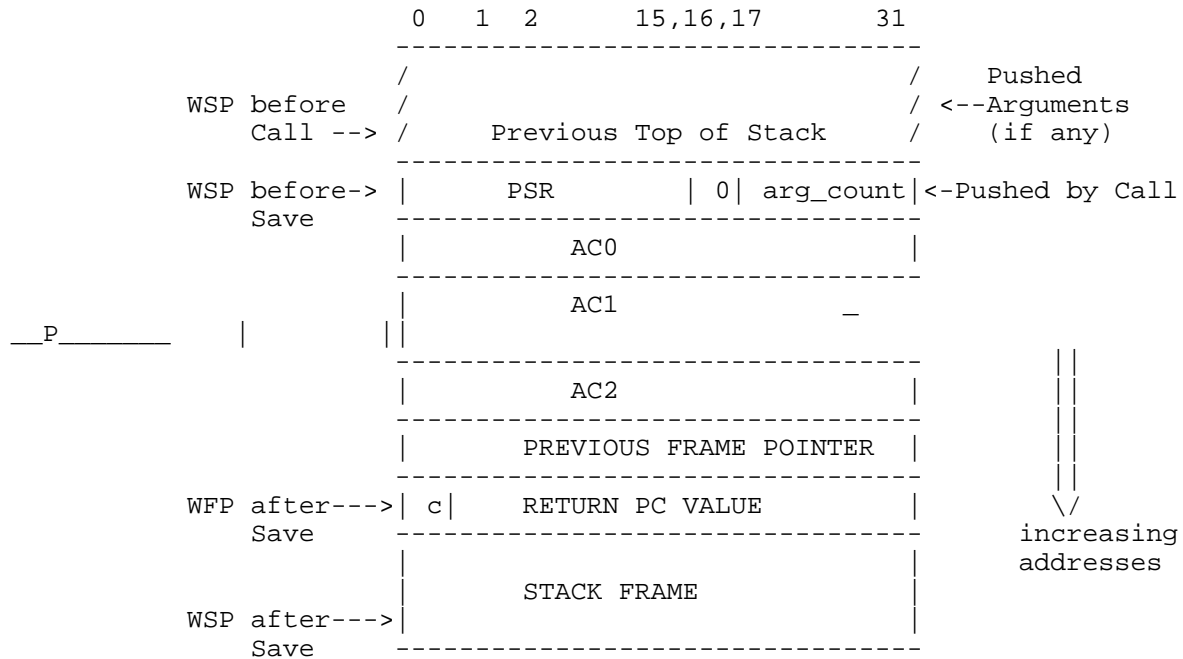


Figure 8-4. Wide Return Block - WSAVR/S

WSAVR - Wide Save/ Reset Overflow Mask

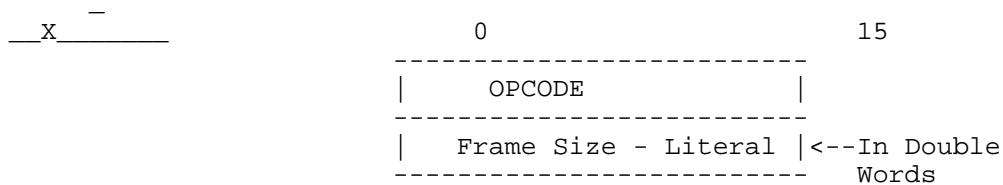
Same as WSAVS except the Overflow Mask is reset to 0, thereby disabling integer overflow.

Data General Corporation  
Company Confidential

Rev. 7

## WSSVS -Wide Special Save/ Set Overflow Mask

Saves the information required by the Wide Return instruction. This instruction is typically executed after a X or LJSR instruction. It should be noted that the X or LJSR can not be used to perform a cross ring call. Also, a call within the same ring, with no arguments can use the X or LJSR & WSSVS/R sequence for a subroutine which uses the WRTN instruction to return to the caller. The format of this instruction is:



A wide return block is pushed onto the stack. After the sixth double word is pushed (the first double word has an argument count of 0), the value of the stack pointer is placed in the frame pointer (WFP) and AC3.

The 16 bit unsigned integer ( the FRAME SIZE ) contained in the immediate field is multiplied by two and added to the stack pointer.

The Overflow Mask(OVK) is set to 1, thereby enabling integer overflow. The Overflow Flag (OVR) is reset to 0.

Before execution, the WSSVS instruction checks for stack overflow. If executing the instruction would result in a stack overflow, none of the above actions occur, and WSSVS transfers control to the stack fault routine. The program counter in the fault return block contains the address of the WSSVS instruction.

The structure of the return block pushed is as follows:

Data General Corporation  
Company Confidential

Rev. 7

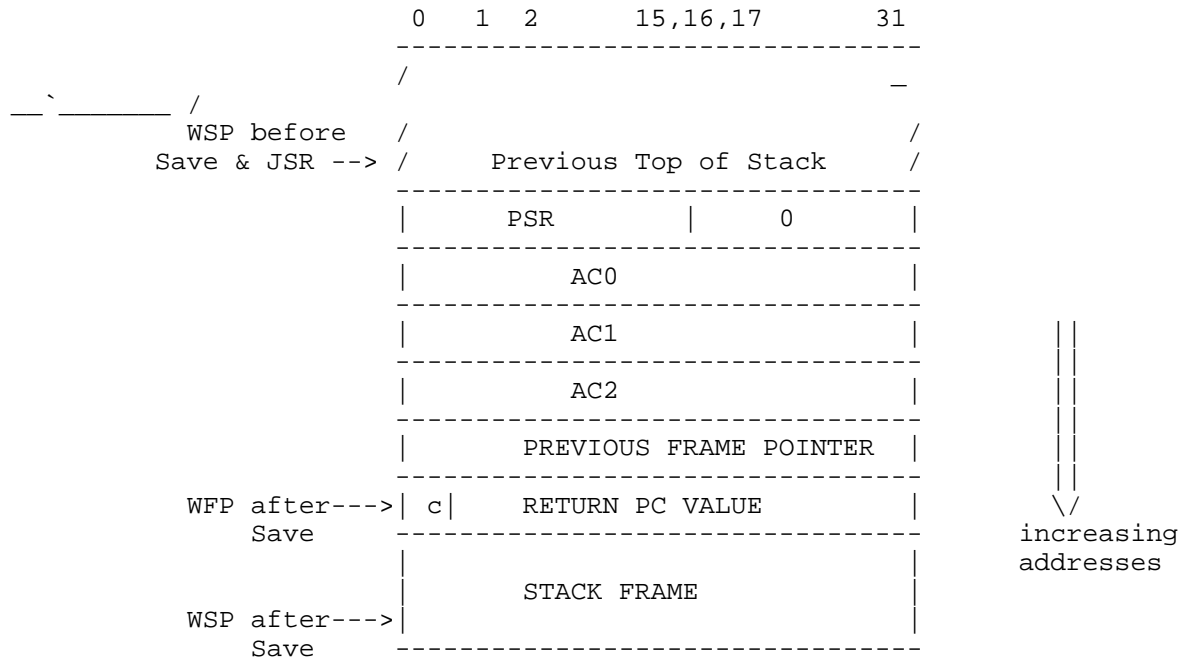


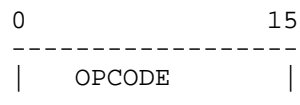
Figure 8-5. Wide Return Block - WSSVR/S

WSSVR - Wide Special Save/ Reset Overflow Mask

Same as WSSVS except the Overflow Mask is reset to 0, thereby disabling integer overflow.

WRTN - Wide Return

The format of this instruction is:



Returns controls from subroutines that issue a WSAV(R or S) or WSSV(R or S) instruction at their entry point. The contents of the frame pointer are placed in the stack

\_\_h\_\_\_\_\_

17/Sep/80

Rev. 7

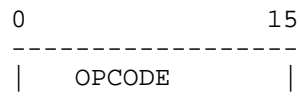
Data General Corporation  
Company Confidential



pointer and a Wide Pop Block Instruction is executed (please see the description of the WPOPB instruction). The popped value of AC3 is placed in WFP. Since the WPOPB subtracted 2 times the arg\_count from the WSP the stack is returned to its state prior to argument pushes.

WPOPB - Wide Pop Block

The format of this instruction is:



Six double words are popped off of the stack and placed in predetermined locations. The double words popped and their destinations are as follows:

Data General Corporation  
Company Confidential

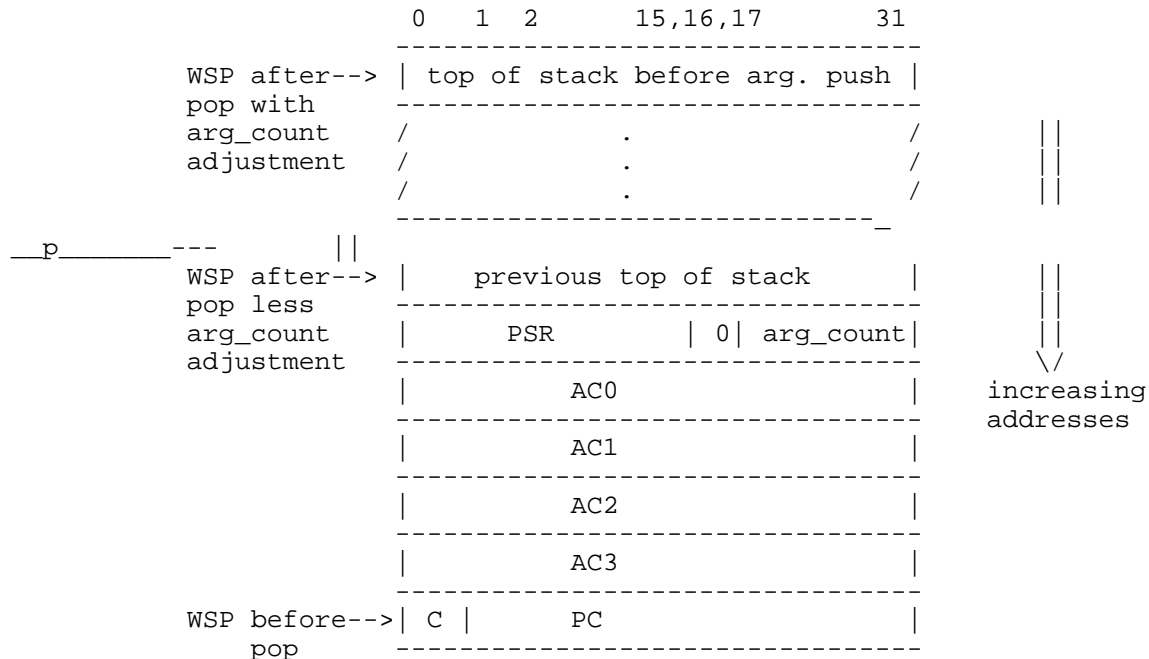


Figure 8-6. Wide Return Block - Before/After Popping

1) If an inward ring crossing is specified, a protection fault is signalled (code=8 in AC1). The current stack remains unchanged. The protection fault occurs after internal state is loaded or being loaded from the popped return block. Consequently, the state of PSR, AC0,1,2,3, Carry, and the PC pushed as part of fault processing is undefined. However, after the return block is pushed, the values of AC0 and AC1 are properly loaded with a fault code and the address of the instruction initiating the fault. Otherwise continue

2) If an intra-ring (same ring) proceed with steps 3 and 4, otherwise step 5.

3) The top six words are popped off of the stack and placed in the specified predetermined locations. OVK, OVR, and IRES are loaded with bits 0,1 and 2 of the last double word popped. WSP\_

--- x <-- WSP - (12 + 2\*arg\_count). A check is made for stack underflow. If stack underflow is detected a stack underflow fault is signalled (code=3 in AC1). Otherwise

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

continue.

4) Sequential operation continues with the word addressed by the updated value of the program counter.

If an outward pop is indicated then the following occurs.

5) The contents of the return block are loaded into the appropriate registers and internal flags. WSP is adjusted ( $WSP \leftarrow WSP - (12 + 2 * \text{arg\_count})$ ). A check is made for stack underflow. If stack underflow is detected a stack underflow fault is signalled (code=3 in AC1). Otherwise continue.

6) WSP and WFP are stored in their respective locations in page 0 of the current segment. WSL and WSB are identical to the values contained in their respective page 0 locations.

7) The outward ring crossing is performed.

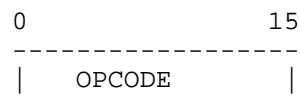
8) The wide stack registers are loaded from their respective locations in page 0 of the new ring.

9)  $WSP \leftarrow WSP - (2 * \text{arg\_count})$ . A check is made for stack underflow. If stack underflow is detected a stack underflow fault is signalled (code=3 in AC1). Otherwise continue.

10) Sequential operations continues with the new value of the PC.

#### WRSTR - Wide Restore

— The Wide Restore instruction is used to return from an interrupt. This instruction has the following format:



This instruction functions in the following way:

Data General Corporation  
Company Confidential

Rev. 7

1) A check is made to see if an inward ring crossing is specified. If it is, a protection fault is signalled (code=8 in AC1). The restore instruction is not executed and the PC in the fault block pushed points to the next instruction. Otherwise continue to step 2).

2) The return block on top of the stack is popped with the appropriate update of the specified registers.

3) The stack management parameters (collectively the stack registers and stack fault address) are popped and temporarily loaded into processor state.

4) If a outward ring crossing is not specified (a restore to the same ring), the following actions occur (steps 5 and 6).

5) The stack management parameters temporarily saved are loaded in the four wide stack registers. The stack fault address temporarily saved is stored into location 14 (wide Stack Fault Address) of the current segment. WSB and WSL are also stored i\_

\_\_\_\_\_nto their page zero locations.

6) Sequential operation continues with the instruction word address by the newly loaded value of the program counter.

7) If an outward ring crossing is specified, the following actions occur.

8) The stack management parameters temporarily saved are stored into the appropriate page 0 locations of the current segment.

9) An outward cross ring restore is performed.

10) The 4 wide stack registers are loaded from the appropriate page 0 locations of the new ring.

11) Sequential operation continues with the instruction word addressed by the newly loaded value of the program counter.

The following is the structure of the stack when a WRSTR instruction is executed.

Data General Corporation  
Company Confidential

Rev. 7



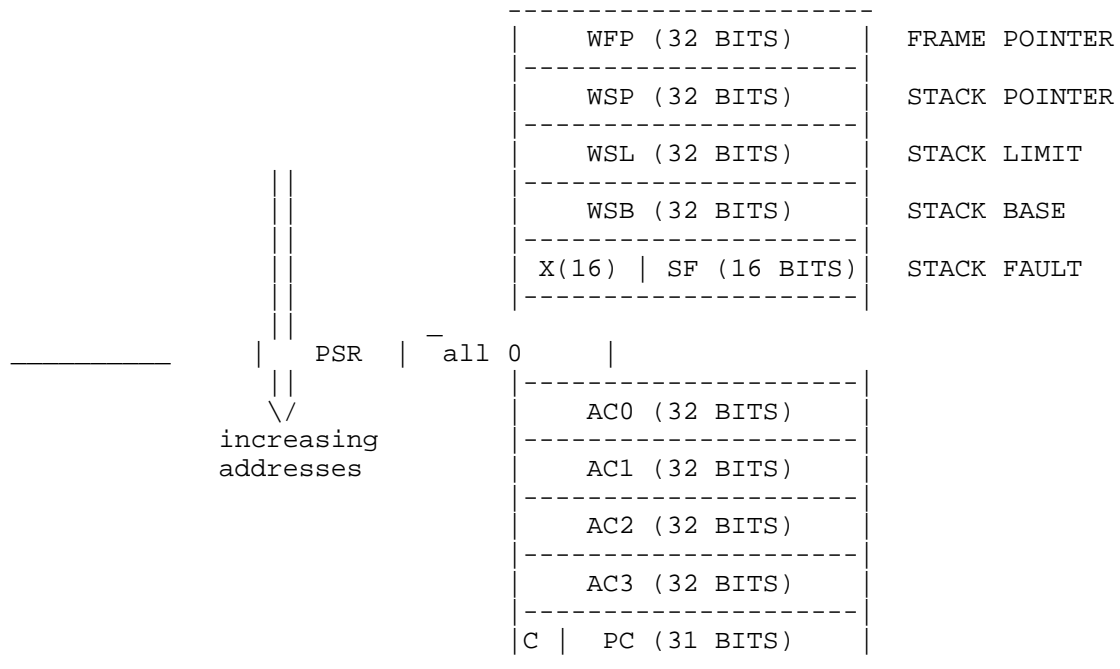
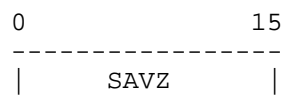


Figure 8-7. Wide Restore Block - WRSTR

SAVZ - Narrow Save with zero frame .

The format of this instruction is:



The Eclipse Narrow Save is performed with an assumed frame size of zero. Please see the description of the Eclipse Narrow save.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 9

### QUEUE INSTRUCTIONS

This chapter describes a set of Eagle instructions which manipulate Queues. A Queue is a linear linked list for which all insertions are typically made at one end of the list and deletions are typically made at the other end. Insertions onto a Queue are referred to as ENQUEUEs. Deletions from a Queue are referred to as DEQU\_

\_\_\_\_\_EUES. The two ends of the linear list of queue insertions are referred to as the HEAD and TAIL of the Queue. A FIFO Queue can be modelled by Enqueuing entries at one end, the TAIL, and Dequeuing entries at the other end, the HEAD. A queue entry is referred to as a DATA ELEMENT. In order to build this linear linked list of queue entries, a Data Element must have two links. One link contains the effective word address of the next entry in the queue. This link is referred to as the FORWARD link. The other link contains the effective word address of the previous entry in the queue. This link is referred to as the BACKWARD link.

The queues defined in Eagle are double ended, that is there is a Head and a Tail. Consequently there must be a mechanism which indicates that a Data Element is the Head or Tail. A Data Element which contains "-1" (32 1's) in its forward link is the tail of the queue. A Data Element which contains "-1" (32 1's) in its backward link is the head of the queue. A queue with one Data Element finds the forward and backward entries both being "-1". A Data Element typically contains user defined data immediately following or preceeding the forward and backward links. The structure and interpretation of this data is left to the user. Defining the head and tail of the queue is a QUEUE DESCRIPTOR. The Queue Descriptor contains two word addresses. One address points to the head of a queue. The other entry points to the tail of the same queue.

Within Eagle, queues are PRIORITY STRUCTURED. This means that enqueues and dequeues need not be at the head or tail of the queue as defined by a Queue Descriptor. Consequently, when a modification to a queue is performed, the queue descriptor and the address of a data element must\_

\_\_\_\_\_ be specified. Enqueues and dequeues are relative to the specified data element.

The following illustrates the structure of the various queue elements defined above and the structure of a queue after various enqueues have been performed.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

The following is a Queue Descriptor referencing an empty queue.

|       |           |
|-------|-----------|
| 0     | 31        |
| ----- |           |
|       | -1 / HEAD |
| ----- |           |
|       | -1 / TAIL |
| ----- |           |

Figure 9-1. Queue Descriptor - Empty Queue

The following is the result of enqueueing a data element located at location A.

|    |                  |          |                  |
|----|------------------|----------|------------------|
|    | 0                | 31       |                  |
|    | -----            |          |                  |
|    |                  | A / HEAD |                  |
|    | -----            |          |                  |
|    |                  | A / TAIL |                  |
|    | -----            |          |                  |
|    | QUEUE DESCRIPTOR |          |                  |
|    | 0                | 31       |                  |
|    | -----            |          |                  |
| A: |                  | -1       | <--Forward Link  |
|    | -----            |          |                  |
|    |                  | -1       | <--Backward Link |
|    | -----            |          |                  |
|    | DATA ELEMENT     |          |                  |

Figure 9-2. Queue with one Data Element

—(—————

Rev. 7

Data General Corporation  
Company Confidential

The following is the result of enqueueing a data element located at location B at the head of the queue before the data element located at A.

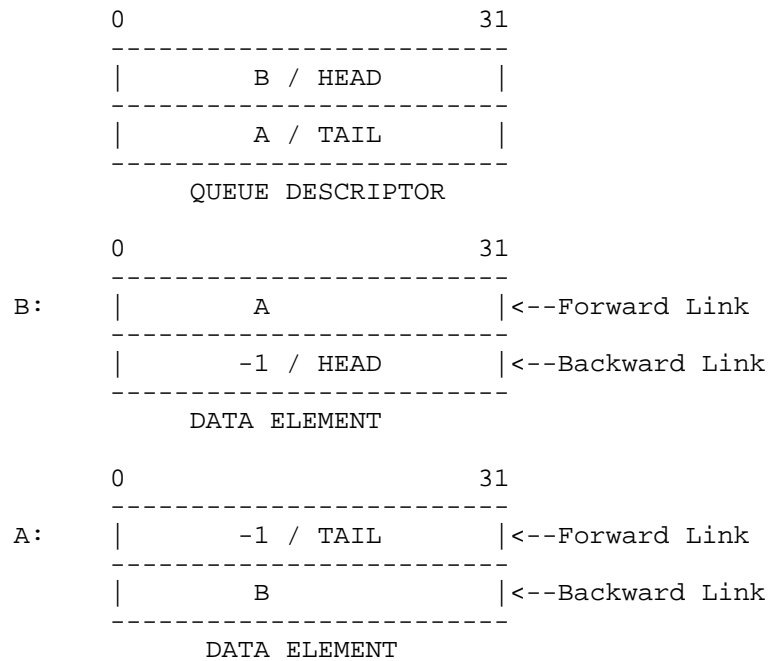


Figure 9-3. Queue with two Data Elements

Data General Corporation  
Company Confidential



And finally enqueueing a data element located at C at the tail of the queue, after data element A.

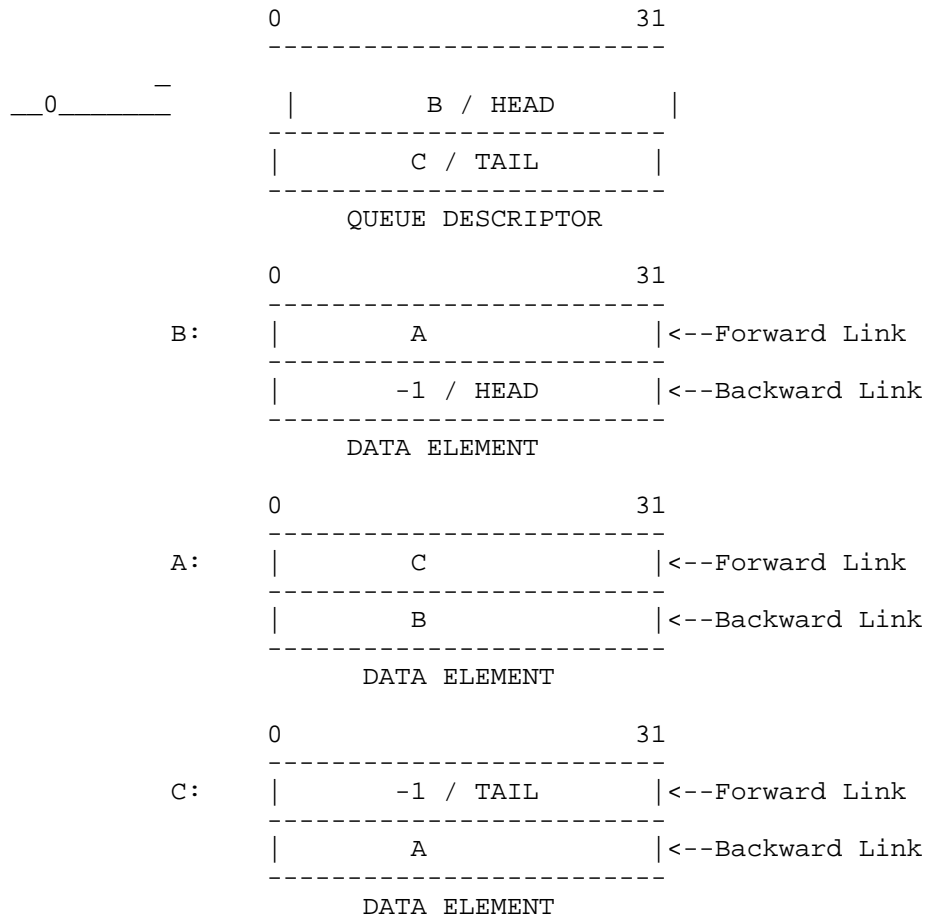


Figure 9-4. Queue with three Data Elements

Data General Corporation  
Company Confidential

Rev. 7

From these examples some of the characteristics of queue manipulations are:

- 1) The Tail of the queue has a -1 in its forward link.
- 2) The Head of the queue has a -1 in its backward link.
- 3) Whenever a new head or tail data element is inserted, the queue descriptor is updated.
- 4) The Forward Link is c\_  
\_\_8\_\_\_\_\_ontained in the first double word of a  
data element.
- 5) The Backward Link is contained in the second double word of a data element.
- 6) An empty queue is indicated by -1 in the head and tail pointers contained in the queue descriptor.
- 7) The backward link references the double word in the previous data element containing the forward link.

The following are the definitions of Eagle instructions which deal with queues that have the structure as illustrated above.

Data General Corporation  
Company Confidential

Rev. 7



Data General Corporation  
Company Confidential

Rev. 7

and backward links of a data element totally contained with  
1 page.

The enqueue operation is not interruptable. The entire  
instruction is completed before any further interrupts are  
enabled. AC0, AC1, AC2 and AC3 are unchanged after the  
enqueue is completed. Carry<-Carry and Overflow<-0.

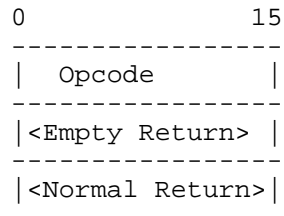
Data General Corporation  
Company Confidential

Rev. 7



## ENQT - Enqueue towards the Tail

The format of this instruction is:



AC0 contains the effective address of a Queue Descriptor. AC1 contains an effective address of a Data Element in the Queue defined by AC0. AC2 contains the effective address of the Data Element to be added to the Queue. The Data Element referenced by AC2 is enqueued (inserted) after the Data Element referenced by AC1. The Queue Descriptor is updated if the enqueue is to the tail of the queue. If the Queue Descriptor indicates that the queue is empty, the contents of AC1 are ignored, and the next sequential word is executed. If the queue is not empty (be\_

\_\_P\_\_\_\_fore the enqueue), the next word is skipped.

Before any modifications are made, a check is made to ensure that the page(s) containing the new data element (referenced by AC2) has read and write access privileges. If it doesn't, the appropriate protection fault occurs (first read and then write), and the queue is unaltered. If the page(s) containing the new data element have valid access privileges, the specified queue operation is performed. It is assumed that once the access privileges of a data element are validated, they do not change. The data element links are updated in a manner such that once a page containing a link is referenced, that link is totally updated before a different data element or queue descriptor is referenced. Consequently 1 resident page is sufficient to completely allow the execution of this instruction.

All reads and writes of links in data elements and in the queue descriptor are checked against the current ring. The ring number of the link addresses must be greater than or equal to the current ring.

It is desirable for performance reasons to have the forward

Data General Corporation  
Company Confidential

Rev. 7

and backward links of a data element totally contained with  
1 page.

\_\_\_\_\_X\_\_\_\_\_ The enqueue operation is not interruptable.\_  
The entire  
instruction is completed before any further interrupts are  
enabled. AC0, AC1, AC2 and AC3 are unchanged after the  
enqueue is completed. Carry<-Carry and Overflow<-0.

Data General Corporation  
Company Confidential

Rev. 7



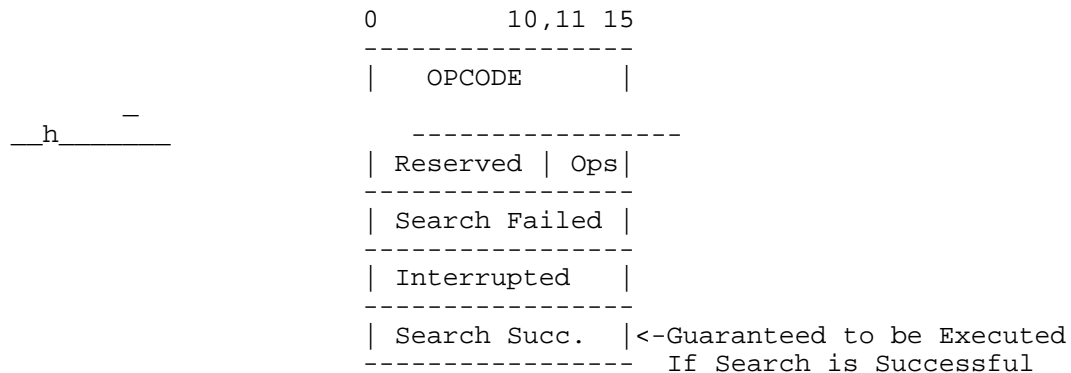
Data General Corporation  
Company Confidential

Rev. 7

The following instructions are used to search a queue for an element containing a status word which meets a specified condition when compared to a mask, and skip if the condition is met. There are 8 conditions, 2 search directions (forward and backward), and 2 mask sizes (narrow and wide). Consequently there are 32 search queue instructions.

(N/W)(F/B)S(SS/SC/AS/AC/E/GE/LE/NE) -

The format of a search queue instruction is:



The second word of this instruction is used to denote which one of the 32 search queue options are desired. Bits 11-15 of this second word, in particular, denote the option. Bits 0-10 are hardware reserved.

This instruction searches a queue for the first data element whose status word, when compared to a mask word, meets the condition specified. The specifiable conditions are:

- 1) Some of the bits specified by the mask are set (SS) in the status word. A "1" in the mask indicates a specified bit.
- 2) Some of the bits specified by the mask are cleared (SC) in the status word. A "1" in the mask indicates a specified bit.
- 3) All of the bits specified by the mask are set (AS) in the status word.
- 4) All of the bits specified by the mask are clear (AC) in the status word.

Data General Corporation  
Company Confidential

Rev. 7



5)The mask and the status word are equal (E).

6)The mask is greater than or equal to (GE) the status word.

7)The mask is less than or equal to (LE) the status word.

8)The mask is not equal to (NE) to the status word.

The values in the mask and status words are unsigned integers for the E/NE/GE/LE arithmetic conditons.

AC1 contains the effective address of the first queue Data Element to be begin the search at. AC3 contains the 2's complement word offset relative to the memory location containing the forward link, that contains the status word. The double word located at the top of the Eagle Stack (referenced by ESP) contains the mask word. If the search specifies a narrow search, then bits 16-31 of the top of stack double word contains the mask and AC3 specifies the relative word offset to the 16 bit status word. If the search specifies a wide search, then the top of stack double word contains the mask and AC3 specifies the relative word offset to the 32 bit status word.

If the search fails, then the next sequential word is executed and AC1 contains the effective address of the last Data Element searched. If the search is successful, two words are skipped and the third word is executed. This third word is guaranteed to be executed. Thus an interrupt can not be taken before the execution of this third word. An interrupt can be taken, however, for the other two returns (i.e., before excuting the word at the return location. In this case, the PC saved by the interrupt handler references the return location, i.e., Search Failed or Interrupted). AC1 contains the effective address of the Data Element that caused the search to be successful.

—  
x  
skipped

If the search is interrupted, then the next word is

and the second word is executed. AC1 contains the effective address of the next data element to be examined. This allows software to make the decision whether to restart the search instruction, pick-up where it left off, or forget the instruction.

For all returns, ESP, AC0, AC2 and AC3 are unchanged. AC1

10:3:27  
17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

is altered as specified above.

A protection or a page fault that occurs during the search is handled as specified in the Fault Chapter. There is no unique return defined for these faults in the search instructions. Carry<-Carry and Overflow<-0.

--End of Chapter--

Data General Corporation  
Company Confidential

## Chapter 10 FLOATING POINT INSTRUCTIONS

This chapter describes the Eagle Floating Point arithmetic algorithms and the instruction set.

### 10.1 Floating Point Arithmetic Algorithms

Eagle and Eclipse floating point algorithms have the following characteristics.

---

All inputs are assumed to have the following structure unless otherwise noted by an instruction: 1)Inputs are normalized, 2)Inputs with a zero mantissa are assumed to also have a zero sign bit and an all zero exponent (TRUE ZERO). An input which does not adhere to these structures results in an undefined output. An input which adheres to this structure is a valid input. An input which has a zero mantissa, but is not a TRUE ZERO is referred to as a DIRTY ZERO. Different implementations may give different results for such an input.

All single precision operations which specify accumulators fetch the most significant 32 bits of the FPAC and ignore the least significant 32 bits. Upon completion of the specified operation the result is returned to the most significant portion of the FPAC, and the least significant 32 bits of this FPAC are set to all 0's.

All floating point instructions leave Carry unchanged and load OVERFLOW with 0, thereby leaving OVR unchanged.

When the mantissa of the result of a computation with valid inputs is all zero, then a TRUE ZERO is generated. More so, computations with valid inputs always generate valid outputs.

#### 10.1.1 Floating Point Rounding

Data General Corporation  
Company Confidential

Rev. 7

## 10.1.1.1 Truncation

When bit 8 of the floating point status register is a 0, 1 zero guard digit is appended to the least significant bit of the mantissa prior to manipulation. All single precision operations manipulate a 28 bit mantissa during intermediate calculations. All double precision operations manipulate a 60 bit mantissa during intermediate calculations.

The results of the intermediate calculations, after post-normalization if required, are truncated to a 24 or 56 bit mantissa for single and double precision operations respectively prior to storage into the specified destination.

## 10.1.1.2 Unbiased Round

When bit 8 of the floating point status register is a 1, 2 zero guard digits are appended to the least significant bit of the mantissa prior to manipulation. All single precision operations manipulate a 32 bit mantissa during intermediate calculations. All double precision operations manipulate a 64 bit mantissa during intermediate calculations.

The results of the intermediate calculations, after post-normalization if required, are rounded to a 24 or 56 bit mantissa for single and double precision operations respectively prior to storage into the specified destination.

The rounding algorithm, "Unbiased Round", functions as follows. The two guard digits (taken as one 8 bit quantity) are examined. If their binary value is 0,1,2,...,127 the post normalized intermediate 24 or 56 bit result is the final result. If their binary value is 129,130,131,..., or 255 the post normalized intermediate 24 or 56 bit result is incremented by 1 to produce the final result. If their value is exactly 128, the least significant bit of the 24 or 56 bit intermediate result is added to the 24 or 56 bit intermediate result to produce the final result. This, in effect, forces the least significant mantissa bit of the result to a "0".

Certain floating point instructions do not use Unbiased Round in their manipulations, even though bit 8 of the FPSR is a 1. Please see the section on Fix/Float Conversions - Integerize in

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



this chapter for a complete discussion on this subject.

## 10.2 Floating Point Indexed Address Instructions

The EAGLE instruction set provides for floating point arithmetic which references memory to have a 31 bit displacement (L). These instructions may be with single or double precision floating point arithmetic and have the following format:

|       |                   |
|-------|-------------------|
| 0     | 15                |
| ----- |                   |
| INDEX | FPAC              |
| ----- |                   |
| @     | DISP BITS 1 - 15  |
| ----- |                   |
|       | DISP BITS 16 - 31 |
| ----- |                   |

\*

LFLDS Floating load single. The memory operand is moved to the specified FPAC. No normalization or cleaning of a dirty zero occurs. If the data is a valid input, N and Z are properly set to reflect the loaded value. If the data is not valid, the values of N and Z are model dependent.

\*

LFLDD Floating load double. The memory operand is moved to the specified FPAC. No normalization or clean-

---

ing of a dirty zero occurs. If the data is a valid input, N and Z are properly set to reflect the loaded value. If the data is not valid, the values of N and Z are model dependent.

\*

LFSTS Floating store single. Moves data from the FPAC to memory. No normalization or cleaning of a dirty zero occurs.

\*

LFSTD Floating store double. Moves data from the FPAC to memory. No normalization or cleaning of a dirty zero occurs.

Data General Corporation  
Company Confidential

Rev. 7

- \* LFAMS Floating add with memory single.
- \* LFAMD Floating add with memory double.
- \* LFSMS Floating subtract with memory single.
- \* LFSMD Floating subtract with memory double.
- \* LFMMS Floating multiply with memory single.
- \* LFMMMD Floating multiply with memory double.
- \* LFDMS Floating divide with memory single.
- \* LFDMD Floating divide with memory double.

The EAGLE instruction set provides for floating point arithmetic which references memory to have a 15 bit displacement (E). This is similiar to C350 floating point instructions, except the address calculation is 31 bits, allowing 32 bit index registers and two word indirect locations. These instructions may be with single or double precision floating point arithmetic and have the follow\_

ing format:

|       |                   |
|-------|-------------------|
| 0     | 15                |
| ----- |                   |
| INDEX | FPAC              |
| ----- |                   |
| @     | DISP BITS 17 - 31 |
| ----- |                   |

- \*
  - XFLDS Floating load single. The memory operand is moved to the specified FPAC. No normalization or cleaning of a dirty zero occurs. If the data is a valid input, N and Z are properly set to reflect the loaded value. If the data is not valid, the values of N and Z are model dependent.
- \*
  - XFLDD Floating load double. The memory operand is moved to the specified FPAC. No normalization or cleaning of a dirty zero occurs. If the data is a valid input, N and Z are properly set to reflect the loaded value. If the data is not valid, the values of N and Z are model dependent.

Data General Corporation  
Company Confidential

Rev. 7

- \* XFSTS Floating store single.
- \* XFSTD Floating store double.
- \* XFAMS Floating add with memory single.
- \* XFAMD Floating add with memory double.
- \* XFSMS Floating subtract with memory single.
- \* XFSMD Floating subtract with memory double.
- \* XFMMS Floating multiply with memory single.

- \_\_\_\_(\_\_\_\_\_ \*
- \* XFMMMD Floating multiply with memory double.
  - \* XFDMS Floating divide with memory single.
  - \* XFDMD Floating divide with memory double.

### 10.3 Floating Point Status Register

The following floating point instructions handle the 64 bit floating point status register which has the format:

| 0                            | 1    | 2                           | 3   | 4   | 5  | 6 | 7 | 8   | 9   | 10 | 11,12 | 15   |
|------------------------------|------|-----------------------------|-----|-----|----|---|---|-----|-----|----|-------|------|
| -----                        |      |                             |     |     |    |   |   |     |     |    |       |      |
| ANY                          | OVF  | UNF                         | DVZ | MOF | TE | Z | N | RND | RES | 00 |       | 0111 |
| -----                        |      |                             |     |     |    |   |   |     |     |    |       |      |
| Reserved (set to all 0's)    |      |                             |     |     |    |   |   |     |     |    |       |      |
| -----                        |      |                             |     |     |    |   |   |     |     |    |       |      |
| 0                            | F.P. | PROGRAM COUNTER - BITS 1-15 |     |     |    |   |   |     |     |    |       |      |
| -----                        |      |                             |     |     |    |   |   |     |     |    |       |      |
| PROGRAM COUNTER - BITS 16-31 |      |                             |     |     |    |   |   |     |     |    |       |      |

Where bits 0 - 15 are the floating point status and bits 33 - 63 are a 31 bit PC. When read or written using the C350 FLST and FSST the bits 32 - 48 are ignored. These instructions are indexed address type instructions with the format:

Data General Corporation  
Company Confidential

Rev. 7



10:3:27  
17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



will be pushed. If the ANY bit is a 1, the FPPC portion of the operand pushed will contain the FPPC of the FPSR. No floating point traps occur. The value of the TE bit is a don't care.

#### WFPOP - Pop Floating Point State

The Eagle Floating Point State is popped from the Eagle stack and loaded into the appropriate floating point registers. The following actions result due to the various encodings of the ANY and TE bits of the popped FPSR. If the ANY bit is a 0, the contents of the Floating Point PC (FPPC) after completion of the instruction is a don't care. Different values on an implementation specific basis will be loaded. If the ANY bit is a 1, the contents of the stack location containing the FPPC image will be loaded into the FPPC portion of the FPSR. No floating point traps occur. The value of the popped TE bit is a don't care.

It should be noted that the C/350 equivalent instructions functions in the same manner with respect to the ANY and TE bits, and the treatment of the FPPC and tra\_@\_\_\_\_\_ps.

The following table summarizes the effect that the values of the ANY and TE of the FPSR have on the FPPC and traps (For the loads and pops the newly loaded value of the FPSR is used). ANY is the logical inclusive OR of bits 1-4. ANY is not necessarily a "Real" bit.

| Instruction | ANY=0   | ANY=1      | TE=0&ANY=1 | TE=1&ANY=1 | TE=x&AN=0 |
|-------------|---------|------------|------------|------------|-----------|
| FLST/LFLST  | FPPC<-x | FPPC<-MEM. | No Trap    | Trap       | No Trap   |
| FSST/LFSST  | FPPC<-X | FPPC<-FPPC | No Trap    | No Trap    | No Trap   |
| FPSH/WFPSH  | FPPC<-x | FPPC<-FPPC | No Trap    | No Trap    | No Trap   |
| FPOP/WFPOP  | FPPC<-x | FPPC<-MEM. | No Trap    | No Trap    | No Trap   |

Where x = software don't care (result is model dependent)  
FPPC<-FPPC = FPPC unchanged

#### FTE - Floating Trap Enable.

The floating point trap enable bit (TE) is set to 1. If ANY is also a 1, a floating point trap is initiated. The value of the FPPC references the floating point instruction which caused ANY to become a 1.

Data General Corporation  
Company Confidential

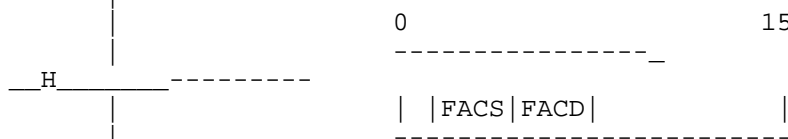
Rev. 7

## 10.4 Conversions: Float/Fix, Integerize, Double/Single

The following instruction rounds a double precision floating point number to a single precision number.

FRDS - Floating-Point Round Double to Single

The format of this instruction is:



Rounds a 64-bit floating point number in FACS to a 32-bit floating-point number and places the result in bits 0-31 of FADC. The rounding is performed using bits 32-63 of FACS as the guard digits. Bits 32-63 effect the result mantissa in the following way:

- \* 0-7FFFFFFF (base 16). Bits 8-31 of FACS become bits 8-31 of the result.
- \* 80000000 (base 16). Bit 31 of FACS is added to bits 8-31 FACS to produce the result mantissa.
- \* 80000001-FFFFFFFF (base 16). 1 is added to bits 8-31 of FACS to produce the result mantissa.

For all rounding the appropriate value of the exponent is maintained, if mantissa overflow occurs. Bits 32-63 of FADC are set to all 0, and the N and Z flags of the FPSR are updated to reflect the newly loaded value of FADC.

NOTE: The rounding algorithm described above always functions in the same manner regardless of the value of bit 8 of the FPSR.

The following floating point instructions reference a fixed point accumulator (AC) and a floating point accumulator (FPAC) and have the format:

Data General Corporation  
Company Confidential

Rev. 7



- \* WFLAD                      Floats from a 32 bit AC
- \* WFFAD                      Fix to a 32 bit AC. N and Z are unchanged.

The following is the algorithm used for float to fixed conversions. For all cases, the contents of the specified FPAC and status bits in the FPSR are unchanged.

- 1) If the conversion is to a 16 bit integer, the floating point number contained in the specified FPAC is effectively adjusted to have an exponent of +4 (without the bias). If the conversion is to a 32 bit integer, the floating point number in the specified FPAC is effectively adjusted to have an exponent of +8 (without the bias).
- 2) If the floating point number contained in the FPAC can not be effectively adjusted, then the MOF bit of the FPSR is set to one. The specified destination is loaded with the integer portion (the higher order bits are truncated), of the contents of the FPAC. If the FPAC is negative, the two's complement of the integer portion is loaded into the specified destination.
- 3) If the floating point number contained in the FPAC can be effectively adjusted, then the specified fixed point destination is loaded. The floating point number is TRUNCATED before loading the fixed point destination.

The C350 FNOM (Normalize) is the only instruction which is guaranteed to accept a non-normalized or dirty zero input and properly produce a valid output. The fix to floating point conversion instructions (e.g., WFLAD) properly convert a zero integer\_

\_X\_\_\_\_\_ to  
a True Zero floating point number.

The C/350 instructions FSCAL and FINT always perform TRUNCATION. This occurs even though bit 8 of the FPSR may be a 1.

The Store Float Single instructions (e.g., LFSTS) store the most significant 32 bits of the specified FPAC. The 32 least

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

significant bits are ignored. Thus, the single precision number stored is the TRUNCATED value of the double precision FPAC. The value of bit 8 of the FPSR is a don't care.

#### 10.5 C/350 Instructions

The Eclipse floating point instructions: FLAS and FFAS read from the least significant 16 bits of the specified AC, and store back a 16 bit integer to the least significant 16 bits of the specified accumulator. The most significant 16 bits are undefined upon completion of the execution of FFAS. The most significant 16 bits of the specified AC are ignored for the execution of FLAS. When the following instructions are executed with 32 bit accumulators, the bit positions of the fixed point accumulators are altered to reflect 32 bit as contrasted to 16 bit accumulators. The functionality and the bit positions relative to the least significant bit are not changed, only the absolute bit numbering when the accumulator is modeled as 32 bits. The effected instructions are: FEXP, FRH, and FSCAL.

FEXP - Bits 17-23 of AC0 are placed in bits 1-7 of the specified  
FPA\_  
\_\_\_\_C. Bits 0-16 and 24-31 of AC0 are ignored.

FRH - Places the higher-order 16 bits of the specified FPAC in bits  
16-31 of AC0. Bits 0-15 of AC0 are undefined.

FSCAL - Shifts the mantissa of the floating point number in the  
specified FPAC either right or left, depending upon the  
contents of bits 17-23 of AC0. AC0 is unchanged upon the  
completion of this instruction.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7



## Chapter 11 DECIMAL NUMERICS

### 11.1 Instruction Set

The following 6 instruction instructions are used in decimal editing and numerical arithmetic manipulations. These instructions possess the same functionality as their Eclipse counterparts except 32\_bit AC's are used. When an Eclipse decimal numeric instruction is executed, all addresses are assumed to be in the current segment. The instructions are:

#### WEDIT - Wide Edit

The Eagle and Eclipse Edit instructions use the same Edit subopcodes. The subopcodes fetched for the Eagle Edit are constrained to be in the current segment. Thus, bits 0-2 of the byte pointer in AC0 are always interpreted as the current segment. The Edit subopcodes function in exactly the same manner with one exception. Whenever a stack is referenced or manipulated, the Eagle Stack is used whenever the edit subprogram is entered by executing the E\_

h\_\_\_\_\_agle

Edit (WEDIT) instruction. In particular, when "j" is used to reference characters in the stack, the number on the stack is at address:

$$ESP + 2 + 2*j$$

The operation uses the number at this address as a character count instead of j.

The Store In Stack (DSTK) subopcode functions as follows when the Eagle stack is referenced.

Stores the byte specified by "p0" in bits 24-31 of a double word in the stack. Sets bits 0-23 of the double word that receives "p0" to 0. If the 8-bit two's complement integer specified by "k" is negative, the instruction addresses the double receiving "p0" by  $ESP + 2 + 2*k$ . If "k" is positive the instruction stores "p0" at the address  $ESP + 2 + 2*k$ .

Data General Corporation  
Company Confidential

Rev. 7

The same interpretation of "k" for DSTK holds true for the DDTK subopcode.

The 8 bit two's complement integer specified in the DADI and DASI subopcodes are sign extended to 32 bits prior to the addition to the Destination and Source Indicator respectively. A Protection Fault (code=4 in AC1) occurs, if the Ring number of the Destination or Source Indicator effected is less than the current ring. This fault occurs

\_\_\_\_\_p\_\_\_\_\_ when these indicators are used to reference memory.

WLDI - Wide Load Integer

WSTI - Wide Store Integer

WLDIX - Wide Load Integer Extended

WSTIX - Wide Store Integer Extended

WLSN - Wide Load Sign

Data General Corporation  
Company Confidential

Rev. 7

## 11.2 Legal Commercial Data Types 0 to 5

The following characters are LEGAL for use in the wide and narrow versions of the LDI, LDIX, LSN, and EDIT instructions.

| Data Types 0 and 1, Sign Position |       |       |       |
|-----------------------------------|-------|-------|-------|
| Character                         | Octal | Value | Sign  |
| -----                             | ----- | ----- | ----- |
| blank                             | 040   | 0     | +     |
| +                                 | 053   | 0     | +     |
| {                                 | 173   | 0     | +     |
| 0                                 | 060   | 0     | +     |
| 1                                 | 061   | 1     | +     |
| 2                                 | 062   | 2     | +     |
| 3                                 | 063   | 3     | +     |
| 4                                 | 064   | 4     | +     |
| 5                                 | 065   | 5     | +     |
| 6                                 | 066   | 6     | +     |
| 7                                 | 067   | 7     | +     |
| 8                                 | 070   | 8     | +     |
| 9                                 | 071   | 9     | +     |
| A                                 | 101   | 1     | +     |
| -----                             |       |       |       |
| C                                 | 103   | 3     | +     |
| D                                 | 104   | 4     | +     |
| E                                 | 105   | 5     | +     |
| F                                 | 106   | 6     | +     |
| G                                 | 107   | 7     | +     |
| H                                 | 110   | 8     | +     |
| I                                 | 111   | 9     | +     |
| -----                             |       |       |       |
| -                                 | 055   | 0     | -     |
| }                                 | 175   | 0     | -     |
| J                                 | 112   | 1     | -     |
| K                                 | 113   | 2     | -     |
| L                                 | 114   | 3     | -     |
| M                                 | 115   | 4     | -     |
| N                                 | 116   | 5     | -     |
| O                                 | 117   | 6     | -     |
| P                                 | 120   | 7     | -     |
| Q                                 | 121   | 8     | -     |
| R                                 | 122   | 9     | -     |

x

B

-

102

2

+

Data General Corporation  
Company Confidential

Rev. 7

|

| Data Types 2 and 3, Sign Position |       |       |       |
|-----------------------------------|-------|-------|-------|
| Character                         | Octal | Value | Sign  |
| -----                             | ----- | ----- | ----- |
| +                                 | 053   | NONE  | +     |
| -                                 | 055   | NONE  | -     |

#### Data\_

#### Type 5, Sign Position

| Octal | Value | Sign  |
|-------|-------|-------|
| ----- | ----- | ----- |
| 014   | NONE  | +     |
| 017   | NONE  | +     |
| 015   | NONE  | -     |

| Data Types 0, 1, 2, 3, and 4 non Sign Posit |       |       |
|---------------------------------------------|-------|-------|
| Character                                   | Octal | Value |
| -----                                       | ----- | ----- |
| blank                                       | 040   | 0     |
| 0                                           | 060   | 0     |
| 1                                           | 061   | 1     |
| 2                                           | 062   | 2     |
| 3                                           | 063   | 3     |
| 4                                           | 064   | 4     |
| 5                                           | 065   | 5     |
| 6                                           | 066   | 6     |
| 7                                           | 067   | 7     |
| 8                                           | 070   | 8     |
| 9                                           | 071   | 9     |

| Data Types 5 non Sign Position |       |
|--------------------------------|-------|
| Octal                          | Value |
| -----                          | ----- |
| 000                            | 0     |
| 001                            | 1     |
| 002                            | 2     |
| 003                            | 3     |
| 004                            | 4     |
| 005                            | 5     |
| 006                            | 6     |
| 007                            | 7     |
| 010                            | 8     |
| 011                            | 9     |

--End of Chapter--

Data General Corporation

Rev. 7

Company Confidential

—  
\_\_\_\_\_



## Chapter 12 PRIVILEGED INSTRUCTIONS

How glorious it is and also how painful to  
be an exception.

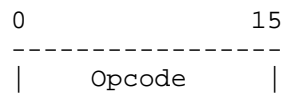
Alfred De Musset.

The following instructions are privileged. They can only be properly executed when the current ring is 0 (i.e., bits 1-3 of the Program Counter are all "0" ). If the current ring is not 0, the execution of these instructions results in a protection fault (Privileged Instruction Violation - code=9 in AC1 ). When the Eagle ATU is enabled, the execution of the Eclipse LMP, SYC, and MMPUL instructions is inhibited and a Privileged Instruction Violation is signalled. For all the privileged instructions, OVERFLOW <-- 0 and Carry <-- Carry. The privileged instructions are:

The VECTOR instruction is a privileged instruction. Please see the chapter on Interrupts for a detailed description of the VECTOR instruction.

LSBRA - Load Segment Base Registers All.

The format of this instruction is:



The word address in AC0 references the base of an 8 double word block. These 8 doublewords are loaded from ascending word addresses into SBR0 thru SBR7 in that order. Upon completion of the register load, instruction execution continues with the updated PC utilizing the newly loaded contents of SBR0 and the ATU is purged. If the processor was in physical addressing mode prior to the execution of this instruction, Eagle \_

\_\_\_\_\_virtual addressing mode is enabled after the SBR's are loaded. If the newly loaded SBR0 is invalid, virtual translation is disabled and logical

Data General Corporation  
Company Confidential

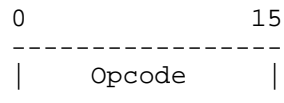
Rev. 7

addresses are identical to physical addresses. The protection fault (code=3 in AC1, due to the invalid SBR0) is now processed. The page 0 locations of segment 0 are in physical memory locations beginning with physical location 0.

At the end of the instruction, AC0 is unchanged.

#### LSBRS - Load Segment Base Registers Some

The format of this instruction is:



The word address in AC0 references the base of a 7 double word block. These 7 doublewords are loaded from ascending word addresses into SBR1 thru SBR7 in that order. Upon completion of the register load, instruction execution continues with the updated PC utilizing the unmodified contents of SBR0 and the ATU is purged of all entries associated with rings 1 thru 7. If the processor was in physical addressing mode prior to the execution of this instruction, Eagle virtual addressing mode is enabled after the SBR's are loaded. If the unmodified SBR0 is invalid, virtual translation is disa\_

\_\_\_\_\_bled and logical addresses are identical to physical addresses. The protection fault (code=3 in AC1, due to the invalid SBR0) is now processed. The page 0 locations of segment 0 are in physical memory locations beginning with physical location 0.

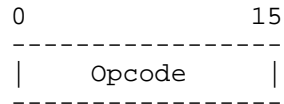
At the end of the instruction, AC0 is unchanged.

#### PATU - Purge Address Translation Unit.

The format of this instruction is:

Data General Corporation  
Company Confidential

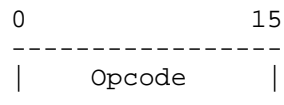
Rev. 7



All the entries in the Address Translation Unit are purged (i.e., marked invalid).

RRFB - Reset Referenced Bit.

The format of this instruction is:



AC1 contains a page frame number in bits 13-31. Bits 28-31 of AC1 are cleared to 0. AC0 contains an origin 0 count (a count of 0 implies 16 page frames). This count indicates the number of words to be reset. The word grouped referenced bits of the page frames beginning at the number specified in AC1 up to and including the word grouped page frames whose number is equal to bits 13-31 of AC1 plus AC0 are reset (loaded with a "0"). If the \_

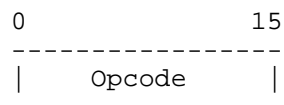
ATU is not enabled,

undefined results will occur. Specification of a non-existent page frame results in an indeterminant data.

At the end of the instruction, AC1 is equal to the original values of AC1 plus AC0 +1. AC0 is all 1's.

ORFB - Or Referenced Bit.

The format of this instruction is:



AC1 contains a page frame number in bits 13-31. Bits 28-31 of AC1 are cleared to 0, forcing the initial pageframe to

Data General Corporation  
Company Confidential

Rev. 7

|

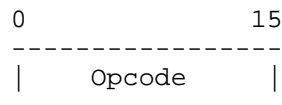
be an integer power of 16. AC0 contains an origin 0 count. This count indicates the number of words to be ORed. The referenced bits of 16 contiguous page frames are grouped for ORing. AC2 contains a word address. The referenced bits of the grouped 16 page frames beginning at the number specified in AC1 are inclusive ORed with a word. This word is part of a word string. The starting address of the word string is contained in AC2. The result of this ORing is stored back in the word string. The referenced bits accessed are reset to "0" upon completion of the ORing operation, AC0 is decremented by 1, AC1 is incremented by 16, and AC2 is incremented by 1. If the ATU is not\_

enabled, undefined results will occur. Specification of a non-existent page frame number results in an indeterminate data.

At the end of the instruction, AC1 is  $AC1 + 1 + 16 \times AC0$ . AC0 is all 1's. AC2 is equal to 1 + the address of the last word used to store the ORed bit string.

LMRF - Load Modified and Referenced Bits.

The format of this instruction is:



AC1 contains a page frame number in bits 13-31. The modified and referenced bits of the specified page frame number are loaded respectively right justified, zero filled into AC0. The referenced bit just accessed is then reset to 0. If the ATU is not enabled, undefined results will occur. Specification of a non-existent page frame number results in an indeterminate data.

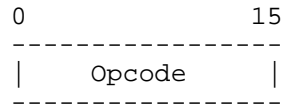
SMRF - Store Modified and Referenced Bits.

The format of this instruction is:

Data General Corporation  
Company Confidential

Rev. 7

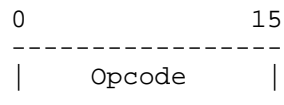




AC1 contains a page frame number in bits 13-31. The least significant two bits of AC0 are stored respectively into the modified a\_0\_\_\_\_\_nd referenced bits of the specified page frame number. If the ATU is not enabled, undefined results will occur. Specification of a non-existent page frame number results in an indeterminate data.

WDPOP - Pop Eagle Context Block.

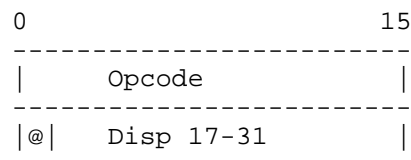
The format of this instruction is:



The information pointed to by locations 32-33 in page zero of segment 0 is used to restore the CPU state to that at the time of the last page fault. Execution of the interrupted program resumes before, during, or after the instruction which caused the fault, depending on the instruction type and how far it had proceeded before the fault.

XVCT - Eagle Vector.

This instruction has the following format:



The displacement field is interpreted as an absolute address in the current segment. Please see the chapter on

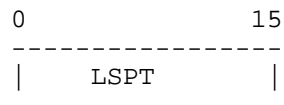
Data General Corporation  
Company Confidential

Rev. 7

\_\_\_\_\_8\_\_\_\_\_on of this interrupt processing for a complete descripti\_ instruction.

LSPT - Load State Pointer.

The format of this instruction is:



This instruction must be executed on implementations that require contiguous main memory allocated to the processor for control purposes. AC0 contains a physical page frame number of the base in memory that the operating system has reserved for the processor. The operating system loads AC0 prior to the execution of this instruction. The length of this area is implementation specific. This instruction must be executed at system initialization time, but before the ATU is enabled. On implementations that do not require reserved physical memory, this instruction causes no operation and need not be executed.

It is possible that other AC's may be used by a particular implementation for other purposes than those defined above. In particular for each implementation that requires that this instruction be executed, the exact functionality expected will be presented in the appropriate machine dependency document.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

# EAGLE I/O INSTRUCTIONS

The following 4 I/O instructions are added to the Eagle Architecture. These instructions permit the implementation of an expanded I/O system. This expansion takes the form of multiple Data General standard Data Channel and Burst Multiplexor busses. This augmentation does not effect controllers nor effect present I/O programs. The Nova P I/O instructions are part of the Eagle Architecture. This expansion is achieved by defining a two level device address; node and device attached on the node. As previously implied, the interfaces between the node and its devices utilize standard Data General interfaces.

The definition of these 4 I/O instructions necessitates the redefinition of certain parts of the Eagle Vector Instruction. When these 4 Eagle instructions are executed, the following actions with respect to protection occurs (when the Eagle ATU is enabled). The WLMP is a privileged instruction. Thus, the current ring must be 0 or a protection fault (code=9 in AC1) is signalled. The other three Eagle instructions ( PIO, CIO, and CIOI) functions as follows. If bit 3 of the SBR of the current ring is a 1, these three Eagle I/O instructions are executed. If bit 3 of the SBR of the current ring is a 0, an I/O Protection Violation (code=10 in AC1) is signalled and the Eagle I/O instructions are not executed. With respect to the Eclipse I/O instructions when the Eagle ATU is enabled the following occurs. If bit 2 of the SBR of the current ring is a 0, the Eclipse I/O instructions may be executed. Bit 3 of the same SBR is then examined. If bit 3 is a 1, the Eclipse I/O instruction is executed. If bit 3 is a 0, an I/O Protection Violation (code=10 in AC1) is signalle\_

H\_\_\_\_\_d. If bit 2 of the SBR is a 1,

the Eclipse LEF is executed. Bit 3 of the SBR is a don't care. The following table summarizes these action.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

| SBR of Current Ring |       |       |                                                                                  |
|---------------------|-------|-------|----------------------------------------------------------------------------------|
| Instruction         | Bit 2 | Bit 3 | Action                                                                           |
| Eclipse I/O         | 0     | 1     | Execute Eclipse I/O Inst.                                                        |
|                     | 0     | 0     | I/O Protection Violation                                                         |
|                     | 1     | x     | Execute Eclipse LEF                                                              |
| Eagle I/O           | x     | 1     | Execute Eagle I/O Inst.                                                          |
|                     | x     | 0     | I/O Protection Violation                                                         |
| WLMP                | x     | x     | Privileged Instruction<br>Current Ring must be 0                                 |
| LMP-Eclipse         | x     | x     | When Eagle ATU is enabled<br>Privileged Instruction<br>Violation (code=9 in AC1) |

x - Hardware Don't Care

These Eagle I/O instructions allow configurations of the type described in the I/O chapter to be controlled. Relative to this configuration, three nodes are present: CPU and first channel, I/O Processor, and Multi-P\_

P\_rocessor Link. The first Channel node is always integral to the CPU with respect to node allocations (IOC - I/O Channel). Commands from one node to another are transmitted over a bus called the, "I/O System Bus (IOSB)". These 4 additional instructions are used to initiate transmission of these commands. Data that is transferred as a result of these commands use the I/O Port Data Bus. For example, an Eagle IOP initiates a command protocol to a disk controller attached to the BMC node. Once the control sequence is initiated, the disk block requested (assume a read) is transmitted over the I/O Port directly to the memory of the IOP. The cache is not involved, nor affected, by the transfer. In fact, concurrently with this I/O transfer, a Processor data request to the main memory system can be occurring.

Referencing an non-existing node is an error. The response to this error is implementation dependent.

When the Eagle ATU is not enabled and logical equals physical address, there is no I/O Protection (the same effect as if bits 2 and 3 of the SBR of the current ring were both 0).

Data General Corporation  
Company Confidential

Rev. 7



### 13.1 I/O Maps and WLMP Instruction

The following instruction loads the Data Channel and Burst Multiplexor Channel Maps.

WLMP - Wide Load Map

The format of this instruction is:

|           |       |        |    |
|-----------|-------|--------|----|
|           | 0     |        | 15 |
|           | ----- |        |    |
|           |       | OPCOD_ |    |
|           | ----- |        |    |
| __X_____E |       |        |    |

AC0 contains a double word with the following format:

|       |          |    |     |       |  |            |
|-------|----------|----|-----|-------|--|------------|
| 0     |          | 16 | ,18 | 20,21 |  | 31         |
| ----- |          |    |     |       |  |            |
|       | RESERVED |    |     | 000   |  | MAP SLOT # |
| ----- |          |    |     |       |  |            |

Map slots 0-1777 (octal) refer to BMC slots. Map slots 2000-2177 (octal) refer to DCH slots.

WLMP is a privileged instruction.

The double word contained in AC0 references the first Map Slot in the specified I/O channel that is to be loaded. AC1 contains a 16 bit unsigned count of the number of map slots in the I/O Channel referenced by AC0 to be loaded. Bits 0-15 of AC1 are ignored. AC2 contains the effective address of the first double word to be loaded into the referenced I/O Channel slots. The format of the double words referenced by AC2 is:

|       |                      |   |                 |       |  |    |
|-------|----------------------|---|-----------------|-------|--|----|
| 0     | 1                    | 2 |                 | 17,18 |  | 31 |
| ----- |                      |   |                 |       |  |    |
| V D   | all 0-Hardware Reser |   | Physical Page # |       |  |    |
| ----- |                      |   |                 |       |  |    |

Where:

V - Valid. The encoding 0 implies valid. The encoding 1 implies access denied.

D - Data. The encoding 0 implies transfer data. The encoding 1 implies transfer zeros.

Data General Corporation  
Company Confidential

Rev. 7

Bits 2-17. These bits must be all 0. Use of these bits are hardware reserved.

Under control of AC0, AC1, and AC2 successive double words from memory are loaded into successive map slots. AC0 references the first map slot to be loaded.

For each map slot loaded, AC0 is incremented by 1, AC1 is decremented by 1 and AC2 is incremented by 2. Upon completion of the WLMP instruction, AC1 contains a 0 in the least significant 16 bits and AC2 contains the address of the word following the last double word loaded. If AC1 is initially zero, no operation is performed.  
Carry<--Carry and Overflow<--0.

Data General Corporation  
Company Confidential

Rev. 7

The effect of the setting of the V and D bits and the direction of the transfer are as follows:

| V - Valid | D - Data | Transfer      | Action                                     |
|-----------|----------|---------------|--------------------------------------------|
| 0         | 0        | From I/O Port | Transfer Data                              |
| 0         | 1        | From I/O Port | Transfer 0's-Node Supplied                 |
| 1         | X        | From I/O Port | Flag Error to device.<br>Transfer aborted. |
| <hr/>     |          |               |                                            |
| h         | 0        | To I/O Port   | Transfer Data                              |
|           | 0        | To I/O Port   | Transfer 0's-Node Supplied                 |
|           | 1        | To I/O Port   | Flag Error to device.<br>Transfer aborted. |

where from I/O Port implies (typically) memory to device and to I/O Port implies device to memory.

#### NOTES:

1)0's are transferred to either a Data Channel or Burst Multiplexor Device.

2)An error detected as a result of an invalid map entry is flagged to the active BMC requesting device. An error detected as a result of an invalid map as a result of an active Data Channel Device results in bit 4 of the IOC(I/O Channel) status register being set to 1.

3)The IOC status register (reg 6000 <octal> of the IOC) is a superset of the BMC status register.

Data General Corporation  
Company Confidential

Rev. 7



Data General Corporation  
Company Confidential

Rev. 7



- \* Bits 1/10 - Reserved.
- \* Bit 11 - IPORT Memory Error.
- \* Bit 12 - Alive - Referenced an non-existant node or an invalid IPORT transfer.
- \* Bit 13 - Always 1.
- \* Bit 14 - IOC Mask Bit.
- \* Bit 15 - IOC is attempting to interrupt.

Register 7701<8> is write only. A system reset, resets to 0 all IOC Masks. The format of Register 7701<8> is:

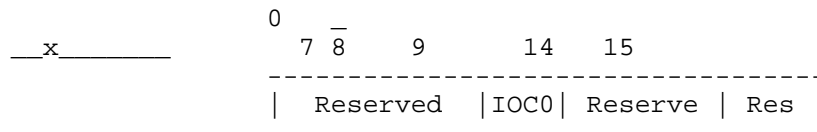


Figure 13-4. IOC Register - 7701<8>

Where:

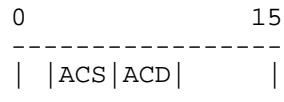
- \* Bits 0/7 - Reserved.
- \* Bit 8 - IOC 0 Mask Bit.
- \* Bit 9 - Reserved
- \* Bit 10 - Reserved
- \* Bit 11 - Reserved
- \* Bit 12 - Reserved
- \* Bit 13 - Reserved
- \* Bit 14 - Reserved
- \* Bit 15 - Reserved.

Data General Corporation  
Company Confidential

Rev. 7

## CIO - Command I/O

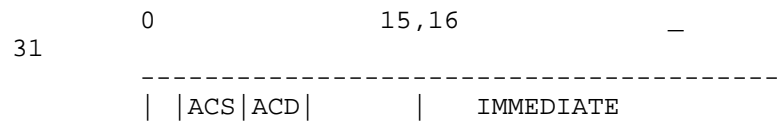
The format of this instruction is:



Bits 17-31 of ACS are a command to be directly issued on the IOSB, with bit 0 of the IOSB forced to a 1. If bit 16 of ACS is a 0, a Read Data operation is then performed on the IOSB and the received data is loaded into bits 16-31 of ACD. Bits 0-15 of ACD are undefined. If bit 16 of ACS is a 1, then a Write Data operation is then performed on the IOSB asserting bits 16-31 of ACD on IOSB DATA. The CIO instruction only issues Format 1 IOSB commands. Carry<--Carry and Overflow<--0.

## CIOI - Command I/O Immediate

The format of this instruction is:



If the ACS and ACD destinations are identical, the immediate field is temporarily saved in internal processor state. If the ACS and ACD destinations are not identical the immediate field is inclusive ORed with the bits 16-31 of ACS and temporarily saved in internal processor state. ACS remains unchanged. Bits 1-15 of this state are a command to be directly issued on the IOSB, with bit 0 of the IOSB forced to a 1. If bits 0 of this state is a 0, a Read Data operation is then performed on the IOSB and the received data is loaded into bits 16-31 of ACD. Bits 0-15 of ACD are undefined. If bit 0 of this state is a 1, a Write Data operation is then performed on the IOSB asserting bits 16-31 of ACD on IOSB Data. The CIOI instruction only issues Format 1 IOSB commands. Carry<--Carry and Overflow<--0.

## PIO -Program I/O

Data General Corporation  
Company Confidential

Rev. 7

The format of this instruction is:

|       |         |
|-------|---------|
| 0     | 15      |
| ----- |         |
|       | ACS ACD |
| ----- |         |

Bits 17-31 of ACS are interpreted as a PIO format IOSB

\_\_\_\_\_command (bit 0 of the IOSB command forced to a 0). The processor will execute the sequence of operations necessary to perform the specified operation, using bits 16-31 of ACD to source or sink data for the transfer. If ACD is a sink, bits 0-15 of ACD are undefined. The PIO instruction only issues Format 0 IOSB commands.  
Carry<--Carry and Overflow<--0.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 14 Interrupts

You shouldn't interrupt my interruptions:  
That's really worse than interrupting.

Thomas Stearns Elliot, "Julia"

An interrupt is an event which occurs asynchronous to the currently executing program. Upon interrupt handling completion, the interrupted program may continue execution.

### 14.1 Interrupt Mechanism

When an interrupt occurs further interrupts are disabled. When the Eagle ATU is enabled, the contents of logical location 1 of page 0 of segment 0 is fetched. This word is interpreted as the address of the interrupt handler. This address is resolved to reference an instruction. One of three possible instruction classes are encountered.

One class is an Eclipse Instruction. Upon detection of an Eclipse instruction the value of the PC pointing to the next instruction that would have been executed \_\_\_\_\_ is stored in location 0 of page 0 of segment 0. Eclipse interrupt processing then proceeds.

Another Instruction class is an Eagle Vector Instruction. When an Eagle Vector instruction is detected, the actions specified in the following section (Vectored Interrupt) are initiated.

The last class is any other Eagle Instruction (ALC with the no-load always skip option) except Eagle Vector, Eagle WBR, or Eagle fixed point memory reference. Eagle WBR and fixed point memory reference (i.e., the Eagle instructions which use the C/350 XOP and XOP1 encodings) are interpreted as an Eclipse instructions for the purposes of initiating the appropriate type of interrupt. When this class of instructions occur the actions specified in the following section (Immediate Interrupt) are initiated.

When the Eagle ATU is not enabled, then if logical addresses are equal to physical addresses all page 0 locations of segment 0

Data General Corporation  
Company Confidential

Rev. 7



are in physical memory beginning with physical location 0. When MMPU1 is enabled, the conventions of page 0 relating to MMPU1 hold. Eagle programs executing with MMPU1 enabled are not supported.

When logical addresses equal physical addresses, and Eagle programs are interrupted, the Eagle Interrupt Mechanism functions as follows. The interrupt functionality described in this section is altered to the extent that all locations described become physical \_

---

addresses. And the rules specified for the generation of physical addresses hold. Please see the section describing physical address generation in the Memory Management chapter.

#### 14.1.1 Immediate Interrupt

When an Eagle instruction other than the Eagle Vector Instruction is referenced by the interrupt handler address in location 1, the following actions are initiated.

Further interrupts are disabled. ESP and EFP are stored in their respective page 0 locations. The program counter value pointing to the next instruction that would have been executed if it were not for the interrupt is loaded into logical locations 2 and 3 of segment 0. A "jump indirect" to the 16 bit interrupt handler address located in location 1 in segment 0 is executed. Execution continues at the instruction referenced by this 16 bit pointer.

When the Eagle vector instruction is executed after an Immediate Interrupt is honored, the following actions occur. If the interrupt occurred at base level, software must load ESB, ESP, EFP, and ESL prior to the execution of the Eagle Vector. If the interrupt occurs at intermediate level, the vector stack is presumed to have been previously established. However, the Eagle Vector pushes the value contained in segment 0, locations 2 and 3. This double word contains the PC value of the interrupted instruction.

#### 14.1.2 Vectored Interrupt

When the Eagle Vector Instruction is referenced by the interrupt handler address in location 1, the following actions are initiated. ( all memory locations are logical ).

— ————. 7

Rev\_

Data General Corporation  
Company Confidential

Word location 0 of segment 0 is fetched. If the contents of this location is 0, base level interrupt processing is initiated. If the contents of this location is not 0, intermediate level interrupt processing is initiated. After this determination has been made, the incremented value of location 0 is restored. The following discussion separately treats base level and intermediate level processing. An epilogue sequence common to all levels of interrupt processing is then presented.

Software, as part of its interrupt return program, decrements location 0.

Locations 2 and 3 are not updated with the value of the program counter pointing to the next instruction when the Eagle Vector Instruction is executed.

In all the following discussion, the value of the PC pushed points to the next instruction of the interrupted program.

#### 14.1.2.1 Base Level

Base level interrupt processing proceeds as a function of the current ring: Ring 0 or any other ring.

##### 14.1.2.1.1 Ring 0-Base Level

1)The 4 stack management registers and fault handler ( ESB, ESP, EFP, ESL, and Eagle Stack Fault Address (location 14<octal>) - collectively referred to as stack management parameters ) are temporarily saved in internal processor state.

2)The 4 stack management registers and stack fault handler address are reloaded from the Vector Stack locations in segment 0. Locations 4 and 6 interpreted as 16 bit word offsets. Thus the vector stack is limited to 128 KB. This mechanism works as follows: ESP, EFP, and ESB are loaded with a zero ext\_

\_\_\_\_(\_\_\_\_\_ended on the left (16 0's)

Vector Stack Pointer (segment 0, location 4). ESL is loaded with a zero extended on the left (16 0's) Vector Stack Limit (segment 0, location 6). Location 14<octal>, the Eagle Stack Fault Address is loaded with the Vector Stack Fault Address contained in word 7. The newly loaded values of WSB and WSL are stored into their appropriate page zero locations. As a result of the zero

Data General Corporation  
Company Confidential

Rev. 7

extension, stack underflow and overflow are enabled when the Vector Stack is initialized.

3)The ring 0 stack parameters temporarily saved in processor state are pushed onto the newly created vector stack.

4)A wide return block, saving the accumulators, overflow flags, and the PC is pushed onto the vector stack.

5)The common epilogue sequence is now performed.

#### 14.1.2.1.2 Non-Ring 0 -Base Level

1)ESP and EFP are restored to their locations in page 0 of the current ring. The values of ESL and ESB in the register set and current ring, page 0 locations are assumed to be identical.

2)A ring crossing to ring 0 is performed.

3)The stack management parameters of ring 0, all contained in segment 0 locations are temporarily saved in internal processor state.

4)Interrupt processing now proceeds as in ring 0 - Base Level after the stack management parameters are temporarily saved in internal processor state. (step 2)

#### 14.1.2.2 Intermediate Level

—0— The following two sections describe interrupt processing when the initial value of location 0 of segment 0 was not 0.

##### 14.1.2.2.1 Intermediate Level - Ring 0

1)A return block, saving the accumulators, overflow flags, and the PC is pushed onto the presently defined ring 0 stack.

2)The common epilogue sequence is performed.

Data General Corporation  
Company Confidential

Rev. 7

## 14.1.2.2.2 Intermediate Level - Non Ring 0

1)ESP and EFP are restored to their locations in page 0 of the current ring. The values of ESL and ESB in the register set and the current ring are identical.

2)A ring crossing to ring 0 is performed.

3)The ESP, EFP, ESL, ESB stack management parameters in ring 0, all contained in segment 0 page 0 locations, are loaded into the appropriate stack registers.

4)Interrupt processing now proceeds as in ring 0 - Intermediate Level. At this time during interrupt processing, the ring 0 stack management parameters have already been initialized to the Vector Stack.

## 14.1.2.3 Common Interrupt Epilogue

Upon the completion of the initial interrupt response, the following common concluding sequence occurs: ( the following table structures are used by this sequence )

\_\_8\_\_poration

Data General Cor\_

Company Confidential



|                                            |         |           |                      |    |  |
|--------------------------------------------|---------|-----------|----------------------|----|--|
| Word 0<br>pointed to<br>by Eagle<br>Vector | WORDS   | 0,1       | 15,16                | 31 |  |
|                                            | -2,-1   | Curr Mask | Must be 0            |    |  |
|                                            | 0,1     | x         | DCT ADDRESS - dev 0  |    |  |
|                                            | 2,3     | x         | DCT ADDRESS - dev 1  |    |  |
|                                            |         | /         | .                    | /  |  |
|                                            |         | /         | .                    | /  |  |
|                                            |         | /         | .                    | /  |  |
|                                            | 126,127 | x         | DCT ADDRESS - dev 63 |    |  |

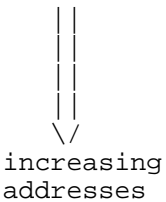

  
increasing  
addresses

Figure 14-1. Vector Table

|       |               |                                    |                      |    |  |
|-------|---------------|------------------------------------|----------------------|----|--|
| WORDS | 0,1           | 7,8                                | 15,16                | 31 |  |
| 0,1   | x             | PC 1-31 (Device Interrupt Routine) |                      |    |  |
| 2,3   | Must be all 0 |                                    | Current Mask         |    |  |
| 4,5   | PSR           |                                    | //////////////////// |    |  |

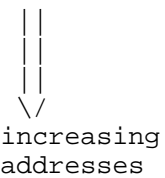

  
increasing  
addresses

Figure 14-2. DCT Entry - First 5 Words

1)The effective address produced by the evaluation of the absolute displacement of the Eagle Vector Instruction references the base of a VECTOR TABLE. The Vector Table contains 64 32\_bit double word entries, one for each device. Each entry references\_@\_\_\_\_\_ the base of a Device Control Table (DCT).

2)The interrupting device # is used as a double word offset from the base of the VECTOR table. The vectored to entry is fetched.

Data General Corporation  
Company Confidential

Rev. 7

3) Bit 1-31 of this vector table entry are used as the address of the first word of the interrupting device's DCT.

4) Place this word address in AC2.

5) Construct a double word as follows: Bits 0-15 are all zero. Bits 16-31 are the contents of the current mask. Push this double word onto the current stack.

6) Place the logical inclusive OR of the double word just pushed and the double word contained in words 2-3 of the DCT in AC0.

7) Proceed to step 8.

8) Place the contents of the least significant 16 bits of AC0 in the Current Interrupt Mask Word.

9) Proceed to step 10.

10) Do a MASK OUT from the least significant 16 bits of AC0 and enable interrupts.

11) Place the 6-bit DEVICE # code of the interrupting device, zero extended on the left to 32 bits in AC1.

12) Place the address of the device interrupt routine ( words 0 and 1 of the DCT ) in the program counter.

11) The contents of word 4, PSR, are used to initialize the OVK, OVR, and IRES flags.

12) Check for stack overflow. If overflow is detected, transfer control to the stack fault handler routine (At this time, the Vector Stack and Vector Stack Fault handler have been previously initialized). If overflow is not detected continu\_

\_\_\_\_\_H\_\_\_\_\_e sequential

operation with the word addressed by the program counter. The first instruction of the interrupt or stack fault handler is executed before further interrupts are honored.

#### NOTE:

1) The addresses contained in the Vector Table and the DCT are not restricted to ring 0. The only rule enforced, is that the pointer chain; interrupt handler, to vector table, and to DCT PC entry must conform to the ring crossing rules as put forth in table 1 of the Protection Chapter. Consequently, the interrupt handler vectored to need not be in Ring 0.

Data General Corporation  
Company Confidential

Rev. 7

## 14.2 Micro Interrupts

From the viewpoint of interrupts, there are three types of instructions:

- 1)Non-interruptable
- 2)Restartable
- 3)Resumable

These types of instructions are now separately discussed.

A Non-interruptable instruction completes all its control sequencing before allowing the processor to honor the interrupt. The value of the PC saved, as part of the processor honoring the interrupt, points to the next instruction to be executed. Instructions like fixed point ADD, SUBTRACT, and LOAD typify instructions of this type. Thus, these instructions never set bit 2 of the PSR.

A Restartable instruction does "NOT" complete all its control sequencing before allowing the processor to honor the interrupt. The value of the PC saved, as part of the processor honoring the

—P\_\_\_\_\_ interrupt, points to this restartable instruction. The programmer visible state, relative to the state at the beginning of the execution of the restartable instruction, is in one of two states: Unchanged or Updated.

Unchanged implies that the entire instruction is restarted when the interrupted program is returned to. Updated implies that the definition of the instruction permitted updating of programmer supplied input parameters in such a way, that when the interrupt was honored these input parameters were updated to reflect the partial completion of the instruction. Floating Point Divide is an example of an Unchanged Restartable instruction. Character Move and Block Move are examples of Updated Restartable instructions. These instructions never set bit 2 of the PSR (Processor Status Register).

A Resumable instruction does "NOT" complete all its control sequencing before allowing the processor to honor the interrupt. The value of the PC saved, as part of the processor honoring the interrupt points to this Resumable instruction. Unlike Restartable instructions, internal processor state (programmer invisible) must be saved in order to resume the instruction. The structure and amount of this internal state is implementation specific. This internal state (micro state block) is pushed onto the current stack

Data General Corporation  
Company Confidential

Rev. 7

prior to the honoring of the interrupt. Then bit 2 of the P\_  
 X\_\_\_\_\_SR is  
 set to 1.

Thus, to use instructions that are micro-interruptable 1)A stack must be defined, and 2)The system interrupt handlers must return to the interrupted program via 1 of 4 instructions which restore bit 2 of the PSR (WPOPB, WRSTR, WRTN, and LPSR). After bit 2 is restored from the appropriate state block or memory location, bit 2 is tested. If it is a 1, the micro state block on the current stack is examined to determine the type of microinterrupt.

As part of this determination, a consistency check is performed on the micro state block. If the micro state block is valid, the instruction is resumed. If the micro state block is invalid, an Eagle protection violation (code=12 in AC1) is signalled. If an Eclipse instruction detects an invalid micro state block, a fault is signalled. The fault type is a function of the type of the Eclipse instruction interpreting the micro state block. If a floating point instruction, a floating point fault occurs. Bit 9 of the FPSR is set to a 1 to indicate an invalid micro state block. If a commercial instruction, an Eclipse commercial fault occurs. Code=5 is loaded into AC1 to indicate an invalid micro state block.

The following is the state of bit 2 of the PSR and bit 9 of the FPSR upon the occurrence of a micro-interrupt for Resumable instructions:

| Instruction           | PSR bit 2 | FPSR bit 9                               |
|-----------------------|-----------|------------------------------------------|
| Eclipse<br>LDI (e.g.) | Set to 1  | Set to 1                                 |
| Eagle<br>WLDI(e.g.)   | Set to 1  | Undefined<br>Implementation<br>dependent |

A\_  
 \_\_\_\_\_ check is performed for stack overflow after the micro state block is pushed onto the current stack. If a stack overflow is detected, the stack fault is honored after the interrupted program is resumed. That is, the interrupt is honored (code=4 in AC1) before the stack fault.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential



When the argument transfer sequence is interrupted upon an inward ring call, the saved PC points to the first instruction of the called procedure (the contents of the referenced GATE).

Upon completion of the return from the micro-interrupt, bit 2 of the PSR is reset to 0.

--End of Chapter--

Data General Corporation  
Company Confidential

## Chapter 15 Fault Mechanism

He who is undetected only gets worse,  
Whereas he who is detected and punished has  
the brutal part of his nature silenced and  
humanized.

"Book IX"  
Plato's Republic

The following faults are defined in Eagle:

\_\_\_\_\_h\_\_\_\_\_ault

- .Stack F\_
- .Protection Fault
- .Floating Point Fault
- .Commercial Fault
- .Page Fault
- .Fixed Point Overflow

The initial pointer to these fault handlers (except page faults) are all 16 bits long. Thus, levels of indirection, if any, assume 16 bit pointers. Levels of indirection, if any, stay within the segment initially containing the pointer.

The first instruction of the Protection and Page Fault handlers is guaranteed to be executed. Thus, an interrupt will not be honored until the completion of this first instruction. For all other faults, an interrupt can occur before the first instruction is executed. In these cases, the PC saved, points to the first instruction of the fault handler. In other words, an interrupt is kept pending until; the return block is pushed, and the PC is updated to reference the first instruction of the fault handler.

Prior to the execution of the fault handler but after the return block is pushed, the OVK, OVR, and IRES (PSR) flags are reset to 0, thereby disabling fixed point overflow faults and clearing microinterrupts.

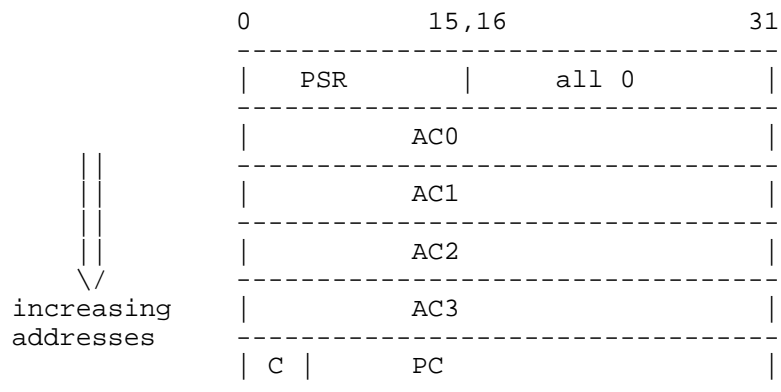
10:3:27  
17/Sep/80

Data General Corporation  
Company Confidential

Rev. 7

After pushing a return block, a check is made for stack overflow. If a stack overflow exists, an additional return block is pushed onto the stack and a stack fault is signalled (code=4 in AC1). In this new return block, the values of AC0, AC1, and    p    PC are the values that were established by the first fault handler. Otherwise the signalling fault is processed.

The structure of the wide return block pushed for all faults (except page faults) is as follows:



The frame pointer is unchanged after the push of the wide return block for those faults which stay within their current segment. The commercial fault handler may push double words in addition to the wide return block.

On implementations with asynchronous floating point units certain interlocks are performed to insure the greatest amount of sequential integrity. These interlocks are described in the floating point fault section of this chapter.

Each of the 6 possible Eagle faults are now separately discussed.

### 15.1 Stack Fault

After every operation that pushes data onto the stack, a check is made for overflow protection. The Eagle stack pointer and the Eagle stack limit are both treated as "unsigned" 32 bit integers and compared. If the stack pointer    x    s greater than the stack

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

limit, a stack overflow condition exists. If a stack overflow condition exists, the processor pushes a wide return block onto the stack with the program counter in the return block pointing to the next logical instruction after the stack instruction that caused the fault. Bit 0 of the stack pointer is reset to 0 and bit 0 of the stack limit is set to 1 (WSL and page zero locations 24-25 <octal>). After the return block has been pushed and bit 0 of the stack pointer and stack limit have been set, the processor executes a "jump indirect" to the stack fault address in location 14 (octal) of the present segment. Stack faults due to stack overflow can be disabled by specifying a stack limit of all 1's.

After every operation that pops data off the Eagle stack, a check is made for underflow protection. The Eagle stack pointer and stack base are both treated as "signed" 2's complement 32 bit integers. If the Eagle stack pointer is less than the Eagle stack base a stack underflow condition exists. If a stack underflow condition exists, the processor sets the Eagle stack pointer equal to the stack limit and pushes a wide return block with the program counter in the return block pointing to the instruction immediately after the stack instruction that caused the fault. Bit 0 of the stack pointer is reset to 0 and bit 0 of the stack limit is set to 1 (WSL and page zero locations 24-25 <octal>). The stack base is unaltered. After the return block has been pushed and bit 0 of the stack pointer and stack limit have been set, the processor executes a "jump indirect" to the stack fault address in location 14 (octal) of the present segment. Stack faults due to stack underflow can be disabled by specifying the most negative signed integer (1 and 31 0's).

For stack underflow, the first double word of the pushed wide return block is at location ESL+2. That is, ESP is initialized to ESL, and ESL is incremented by 2 prior to the pushing of the wide return block.

The following codes are loaded into AC1 when a stack fault is detected:

| CODE | MEANING                                                                                                                                                                                    |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0    | Overflow on every stack operation other than SAVE, WMSP, or ring crossing.                                                                                                                 |
| 1    | Underflow or Overflow would occur if the instruction was executed (WMSP, WSSV(R or S), or WSAV(R or S). The PC in the return block references the instruction that caused the stack fault. |

Data General Corporation  
Company Confidential

Rev. 7



- 2 Too many arguments on a cross ring call.
- 3 Stack Underflow.
- 4 Overflow due to a return block pushed as a result of a microinterrupt or fault.

AC0 is loaded with the address of the instruction which caused the stack fault.

Eagle stack manipulations are implemented using Eagle specific instructions. Thus the stack fault processing mechanism knows that the Eagle fault mechanism is to be invoked. However, jump indirects interpret 16 bit pointers.

## 15.2 Protection Fault

A protection fault occurs due to the specification of an action that is not permitted. When a protection fault is detected the following actions occur:

.A ring crossing to ring 0 is performed if the current ring is not ring 0. ESP and EFP are stored in their respective positions in page zero of the current ring.

.A wide return block is pushed onto the ring 0 stack. PC in this return block points to the next instruction that would have been executed. AC0 contains the address of the instruction which caused the fault.

.A code indicating the type of fault is placed in AC1.

.A "jump indirect" thru the 16 bit pointer contained in logical location 36 of segment 0 is executed.

The codes placed in AC1 and their meanings are as follows:

| CODE | MEANING                              |
|------|--------------------------------------|
| 0    | Read Violation                       |
| 1    | Write Violation                      |
| 2    | Execute Violation                    |
| 3    | Validity Bit Protection (SBR or PTE) |

Data General Corporation  
Company Confidential

Rev. 7

|                |    |                                                 |
|----------------|----|-------------------------------------------------|
|                | 4  | Inward Address Reference                        |
|                | 5  | Defer (indirect) Violation                      |
|                | 6  | Illegal Gate - Out of Bounds or Bracket Compare |
|                | 7  | Out_                                            |
| _____ward Call |    |                                                 |
|                | 8  | Inward Return                                   |
|                | 9  | Privileged Instruction Violation                |
|                | 10 | I/O Protection Violation                        |
|                | 11 | Spare - presently not used.                     |
|                | 12 | Invalid microinterrupt return block             |

Eagle protection faults are specific to the Eagle Address Translation Unit. The protection fault processing mechanism knows that the Eagle protection fault processing mechanism is to be invoked. However, the jump indirect functions on 16 bit pointers in logical space. The processor remains in a logical address mode and never disables the "MAP". This differs from C/350 processing in that C350 pointers are in physical address space and mapping is disabled.

If logical equals physical and a protection fault occurs (presently limited to Invalid Microinterrupt Block ), the indirect chain of 16 bit pointers beginning at location 1 (interrupt handler) is resolved to reference an instruction. If this instruction is an Eagle instruction, an Eagle Protection Fault is initiated. If this instruction is any other instruction, an Eclipse Commercial or Floating Point fault is initiated. Please see the section on microinterrupts in the Interrupt Chapter for a discussion of Eclipse fault handling upon the detection of an invalid microinterrupt state block.

### 15.3 Floating Point Faults

Upon completion of any floating point instruction, if any of bits 1-4 of the Floating Point Status Register , FPSR, are set, a floating point fault is indicated. If bit 5 in the FPSR is also set, a response to the floating point fault is initiated, after the execution of the floating point instruction that caused the fault and before the execution of the next sequential instruction. The

---

Floating Point Program Counter contains the address of the first Floating Point instruction that caused a fault to be indicated.

On implementations with asynchronous floating point units, fixed point instructions which do not depend on the results of a floating point computation may be executed simultaneously.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

Consequently, the value of the Floating Point PC and the PC may not be identical.

#### 15.3.1 Asynchronous Floating Point Units

On implementations with asynchronous floating point units, an interlock is performed for all instructions which can perform a change of an address context (frame pointer or current ring of execution). This is to ensure that a fault, if it occurs, stays within the appropriate address context. The instructions that perform this interlock are: CALL, SAVE (all variations, narrow and wide), RETURN (narrow and wide), POPB (narrow and wide), STAFP, SPSR, RESTORE (narrow and wide), WDPOP, and PBX.

Additionally, the following events must wait for parallel execution units to finish before performing an context switch: Page and Protection Faults. The following events must wait for parallel execution to complete, then perform the floating point trap (if pending), and then perform the original event:

- Commercial Fault
- Fixed Point Fault
- Stack Fault
- Interrupts

A floating point trap will be taken as soon as the existence of the trap is known, and not wait until the next worst case interlock is encountered.

If the array processing instruction set is installed, bits 10 and 11 of the floating point status register are used to specify continue on exponent overflow and underflow. If a continuation is specified, a floating point trap is not initiated. Please see the appendix on the array processing instruction set for a complete description of these trap options.

The indirect chain of 16 bit pointers beginning at location 45 (octal) of the present segment is resolved. If the instruction encountered is an instruction with the no-load/ always skip option specified, an Eagle wide return block is pushed onto the present Eagle stack and the encountered Eagle instruction is executed. Otherwise, an Eclipse return block is pushed onto the Eclipse stack.

The wide return block pushed has the following format:

Data General Corporation  
Company Confidential

Rev. 7

| WORD # | CONTENTS                                        |
|--------|-------------------------------------------------|
| 0,1    | PSR, 16 0's                                     |
| 2,3    | AC0                                             |
| 4,5    | AC1                                             |
| 6,7    | AC2                                             |
| 8,9    | AC3                                             |
| 10,11  | Bit 0 = carry bit.<br>Bits 1-31= return address |

This return address is the address of the next user instruction to be executed. The Store Floating Point Status Instruction (\_\_\_\_n) should be used to determine the address of the floating point instruction that caused the fault.

The trap enable bit, bit 5 in the FPSR, is reset to 0 after the Eagle or Eclipse (wide or narrow) return block is pushed.

#### 15.4 Commercial Faults

##### 15.4.1 WLSN/LDIX/STIX/WLDIX/WSTIX

In the course of processing decimal instructions, the processor performs certain checks on the data being processed. If an invalid data type or number is found, a fault is initiated. When a fault occurs, the processor first pushes a wide return block onto the present stack with the program counter double word in the return block pointing to the instruction that caused the fault. The processor then places a code indicating the type of fault in AC1, and executes a "jump indirect" to the decimal fault address located in location 46 (octal) of the present segment.

For all wide commercial faults, bit 16 of AC1 is set to 1 in addition to the listed codes. For all narrow faults bit 16 of AC1 is set to 0. Both wide and narrow commercial faults use the same decimal fault address in page zero of the current segment.

The following codes are produced for decimal faults:

Data General Corporation  
Company Confidential

Rev. 7



| CODE     | INSTR.                | MEANING                                                        |
|----------|-----------------------|----------------------------------------------------------------|
| 1        | LDIX<br>STIX          | Reserved data type 7.<br>Return Block Type 1                   |
| 4        | WLDI                  | Number too large to convert to specified                       |
| <u>0</u> | —<br>WSTI<br>WSTIX    | data type. Return Block Type 1                                 |
| 6        | WLSN<br>WLDI<br>WLDIX | Sign code is invalid for this data type<br>Return Block Type 1 |
| 7        | WLSN<br>WLDI<br>WLDIX | Invalid digit.<br>Return Block Type 1                          |

The structure of the Eagle Return Block Type 1 pushed for the Decimal Numeric instructions (the Eclipse Return Blocks contain the same information with the exception that 1)No Eclipse PSR is defined, & 2)The AC and PC values are 1 word each) is:

| Word | Contents                                                                              |
|------|---------------------------------------------------------------------------------------|
| 0    | PSR, 16 0's                                                                           |
| 2    | AC0 - Unchanged                                                                       |
| 4    | AC1 - Original Descriptor                                                             |
| 6    | AC2 - Original Source Indicator (Destination Indicator for STI group of instructions) |
| 8    | AC3 - Undefined                                                                       |
| 10   | C,PC - PC of the Decimal Numeric Instruction                                          |
|      | Return Block Type 1.                                                                  |

The accumulators are identical to their respective values in the return block with one exception, AC1 contains the fault code.

#### 15.4.2 EDIT and WEDIT

Data General Corporation  
Company Confidential

Rev. 7

In course of processing an WEDIT instruction, the processor performs certain checks on the data being processed. If the instruction encounters an invalid data type or number, the processor initiates a fault. The processor pushed a wide retur\_

8\_\_\_\_\_n block onto

the Eagle stack with the program counter in the return block pointing to the instruction that caused the fault. The processor then places a code indicating the type of fault in AC1, and executes a "jump indirect" to the WEDIT fault address, location 46 (octal) of the present segment.

On an implementation basis, additional words (double words) may be pushed as a result of EDIT faults that result in codes; 0, 2, or 3 being loaded loaded into AC1. In other words, Return Block Type 2 may be bigger.

Words of the stack may be used by WEDIT as temporary storage (implementation specific). The fault handling routine must be aware of this. When a fault occurs, the return block is pushed after any words that are used. The following table indicates the codes used and their meaning for WEDIT faults.

For all wide commercial faults, bit 16 of AC1 is set to 1 in addition to the listed codes. For all narrow faults bit 16 of AC1 is set to 0. Both wide and narrow commercial faults use the same decimal fault address in page zero of the current segment.

Data General Corporation  
Company Confidential

Rev. 7

| CODE | MEANING                                                                                                                            |
|------|------------------------------------------------------------------------------------------------------------------------------------|
| 0    | An invalid digit or alphabetic character was encountered by one of the character move subopcodes (e.g., DMVA). Return Block Type 2 |
| 1    | Invalid data type ( 6 or 7); Return Block Type 3                                                                                   |
| 2    | DMVA_ or DMVC opcode with source data type 5; Return Block Type 2                                                                  |
| 3    | An invalid op-code was encountered; Return Block Type 2                                                                            |
| 6    | Sign code is invalid for this data type; Return Block Type 3                                                                       |

The structure of the Eagle Return Blocks Type 2 and 3 pushed for the Edit instructions are (the Eclipse Return Blocks contain the same information with the exception that 1)No Eclipse PSR is defined, & 2)The AC and PC values are 1 word each):

| Word | Contents                          |
|------|-----------------------------------|
| 0    | PSR, 16 0's                       |
| 2    | AC0 - Current value of P          |
| 4    | AC1 - Undefined                   |
| 6    | AC2 - Undefined                   |
| 8    | AC3 - Undefined                   |
| 10   | C,PC - PC of the Edit Instruction |
|      | Return Block Type 2.              |

Where the current value of P references the EDIT subopcode that caused the commercial fault. That is, if the Ith. subopcode caused the fault, AC0 contains a byte pointer to the I th. subopcode.

Data General Corporation  
Company Confidential

Rev. 7

| Word                 | Contents                          |
|----------------------|-----------------------------------|
| 0                    | PSR, 16 0's                       |
| 2                    | AC0 - Unchanged                   |
| 4                    | AC1 - Undefined                   |
| 6                    | AC2 - Undefined                   |
| 8                    | AC3 - Undefined                   |
| 10                   | C,PC - PC of the Edit Instruction |
| Return Block Type 3. |                                   |

The accumulators are identical to their respective values in the return block with one exception, AC1 contains the fault code.

Eagle commercial instructions which can cause a commercial fault are specific to Eagle. Thus the commercial fault processing mechanism knows that the Eagle fault mechanism is to be invoked. However, the jump indirect functions on 16 bit pointers.

## 15.5 Page Fault

When an attempt to reference a location which is part of the logical address space but is not part of the physical address space a page fault occurs. A page fault is initiated only after the validity and appropriate access bits in a page table entry are interpreted.

A page fault can also occur as a result of a logical reference which requires a 2 level pagetable, but only a 1 level has been allocated. The following actions occurs when a page fault is detected:

- .A context block (the internal state of the machine, implementation specific ) is stored into memory using the contents of word locations 32-33 (octal) of segment 0 as a base address.

- .If the current segment is not 0, EFP and ESP are stored in their respective positions in page 0 of the current segment. Then a ring crossing to Ring 0 is performed. The Ring 0 stack is initialized from page 0 of Ring 0.

- .A WJMP indirect to locations 30-31 (octal) of segment 0 is

Data General Corporation  
Company Confidential

Rev. 7



\_\_P\_\_\_\_ executed.

A status field in the context block indicates the cause of the page fault. The three causes are: page table depth, page fault when referencing a pagetable, and page fault when not referencing a pagetable.

A page fault must not occur during any of the above actions. If it does the processor halts. Further sequences may be initiated by the processor. This is an implementation specific action.

Once the page has been brought into physical memory, or a 2 level pagetable created, the page fault handler can restart the interrupted program by executing a WDPOP instruction. WDPOP restores the processor state from information in the context block.

## 15.6 Fixed Point Overflow

Upon completion of specific Eagle fixed point instructions, the OVR flag is set. If the OVK is also set at this time, a fixed point fault is signalled. The indirect chain of 16 bit point beginning at location 37 (octal) of the present segment is resolved. A wide return block is pushed onto the present stack and the instruction referenced by the resolved indirect chain is executed.

The following instructions load OVR and OVK as part of their execution; SPSR, XVCT, WPOPB, WDPOP, WRTN, WRSTR, and WSAVS. If the newly loaded values of OVR and OVK are BOTH 1, a fixed point trap IS NOT INITIATED. OVR and OVK remain unchanged after the execution of these instructions unless subsequent instructions are executed which directly alter OVK and OVR. A fixed point overflow DOES NOT occur if an arithmetic instruction producing no OVERFLOW is executed when OVR and OVK are one. Fixed point overflow is initiated only when OVK and OVERFLOW are both one. OVR is ignored.

\_\_X\_\_\_\_ The wide return block pushed has the following format:

|       |                    |
|-------|--------------------|
| 0,1   | PSR, 16 0's        |
| 2,3   | AC0                |
| 4,5   | AC1                |
| 6,7   | AC2                |
| 8,9   | AC3                |
| 10,11 | Bit 0 = carry bit. |

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

Bits 1-31= return address

| In the pushed PSR, OVK is a 1 and OVR is a 0. After the push,  
| OVR is then cleared to 0.

This return address is the address of the next user instruction to be executed. If the OVK is a 0, a fixed point fault is not signalled.

The address of the instruction which caused the fixed point fault is loaded into AC0 after the wide return block is pushed.

Fixed Point Faults occur after the execution of the fixed point instruction that caused the fault and before the execution of the next sequential instruction. Interrupts can occur after the return block is pushed and OVR and OVK are reset to zero.

--End of Chapter--

Data General Corporation  
Company Confidential

|

## Chapter 16 Unimplemented Instructions

Location 15 <octal> of page zero of each ring contains a pointer to a table of 32-bit entries. These entries are used to specify the actions that occur when one of four classes of unimplemented instructions are executed. For the purposes of this feature, an unimplemented instruction is an instruction for which processor control sequences DO NOT exist. These control sequences do not exist for two reasons. The first being that no instruction is defined for the particular opcode encoding. The second being, that for one reason or another processor control sequences do not exist for an architecturally defined instruction.

At the option of the programmer, software can be provided that emulates instructions that do not have processor control sequences or detects the presence of an undefined opcode encoding.

The structure of location 15 and the table it references is as follows:

| 0 | 15      | 0      | 31 |                |
|---|---------|--------|----|----------------|
| N | Address | -----> | N  | UIT0 - Eagle   |
|   |         |        | N  | UIT1 - Eclipse |
|   |         |        | N  | CIS            |
|   |         |        | N  | SIS            |
|   |         |        |    | \\ /           |
|   |         |        |    | incr           |
|   |         |        |    | addr           |

UIT functions in the following manner. There are four UIT traps defined. One trap is for all undefined Eagle instructions. The second trap is for all undefined Eclipse instructions. The third trap is for all eagle ext\_ended commercial instructions (they are not defined yet). The fourth and last is for the Eagle Scientific Instruction Set (SIS).

When the WCIS instruction is executed, the second word of the instruction is loaded into bits 16-31 of AC0. Bits 0-15 of AC0 are undefined. When the WAP instruction is executed, the second word of the instruction is loaded into bits 16-31 of AC0. Bits 0-15 of AC0 are undefined.

17/Sep/80  
Rev. 7

Data General Corporation  
Company Confidential

|

When one of these instruction classes is executed and no control processor control sequences exist then the following occurs. Bit 0 of locations 15 of page zero of the current ring is tested. If bit 0 is a 0, the fetched instruction causes no operation to occur. If bit 0 is a 1, then the appropriate table entry is fetched. The contents of these table entry are interpreted as follows:

|       |      |   |   |         |
|-------|------|---|---|---------|
| 0     | 1    | 3 | 4 | 31      |
| ----- |      |   |   |         |
| N     | Ring |   |   | Address |
| ----- |      |   |   |         |

If bit 0 of this entry (N) is a 0, the fetched instruction causes no operation to occur. If bit 0 of this entry is a 1, an UIT trap occurs. If the RING field of the fetched entry is greater than the current ring a protection fault occurs. If the RING field is equal to or less than the current ring an implicit CALL instruction with no arguments is executed. The effective target\_

p of the  
implicit CALL is bits 1-31 of the fetched entry.

All present undefined Eagle opcodes (including the old Eclipse XOP) use UIT0. All undefined EClipse opcodes use UIT1.

The undefined Nova IO instructions (NIO and SKP for ACS <> AC0) are reserved for implementation specific functions. These instructions do not use UIT and are generally IO protected.

### 16.1 Undefined Opcodes

The following are the undefined Eagle opcodes:

```

1 ACS ACD 10111001001 - IO Protected

100 AC 11001101001
100 AC 11001111001

1xxx111111xx1001 - 5-bit immediate field

11000111001x1001
1110011100011001
1110011100101001
1110011100111001
11100111111x1001

```

Data General Corporation  
Company Confidential

Rev. 7

|



```

1 AC 10111xxxx1001
1 AC AC 1xx11011000

```

The following are the Eclipse undefined opcodes:

```

1111111011101000
1101111011101000
1111011011101000

```

```

1 AC AC 11110001000
1111011111001000

```

1xxxx11100001000 - except xxxx = 0010

```

111 AC 1 XR 01111000

```

```

111 AC 1 XR 01111000
101 AC 1 XR 10111000
110 AC 1 XR 10111000
111 AC 1 XR 10111000
100 01 1 XR 10111000
100 10 1 XR 10111000
100 11 1 XR 10111000
111 AC 11011111000
1xxxx 11111101000

```

Unused NIO and SKP instructions ( when ACS is other than AC0)  
are defined only \_

\_\_x\_\_ on an implementation basis, except as LEFs.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 17

### I/O Specification

This section details the I/O specification for the Eagle computer system. The Eagle processor can communicate with I/O devices using three methods: programmed I/O, data channel I/O, and high speed channel I/O. Eagle is I/O program compatible with the C350. It contains both a standard data channel and a burst multiplexor channel. (Multiple burst multiplexor channels are optional). An I/O processor which is program compatible with the M600 I/O processor is also an option on Eagle.

In addition to the Eclipse I/O instructions enumerated in this chapter, Eagle I/O instructions have been defined. Please see the Eagle I/O instruction chapter.

#### 17.1 Programmed I/O

Eagle executes all Eclipse I/O instructions in exactly the same manner as the C350. The number of Eclipse device codes(64) remains unchanged. For the sake of completeness the Eclipse I/O instructions are tabulated below.

|        |            |
|--------|------------|
| DIA<f> | ac, device |
| DIB<f> | ac, device |
| DIC<f> | ac, device |
| DOA<f> | ac, device |
| DOB<f> | ac, devie  |
| DOC<f> | ac, device |
| SKP<t> | device     |
| NIO<t> | device     |
| INTEN  |            |

Data General Corporation  
Company Confidential

Rev. 7

INTDS  
INTA ac  
MSKO ac  
VCT<@> displacement  
IORST  
HALT

A new Eagle Vector Instruction is defined. This instruction is not an I/O instruction.

XVCT

This instruction is specified in the instruction specification section of this document. It is used to generate and push data onto an interrupt stack.

## 17.2 Data Channel I/O

Data channel devices are controlled in two phases. Phase 1 specifies the parameters of the transfer, i.e., the starting location in memory and the number of words to be transferred. Data channel devices can specify logical addresses upto 17 bits (EXDCH can be considered to be the most significant bit of the logical address). This is done with programmed I/O instructions. Phase 2 consists of either a read or a write command to the device. Once the command is issued, the data transfer takes place when the data channel device is ready. Because data channel transfers go directly to the system cache the CPU is not stopped except when there is conflict for the I/O bus.

When a data channel device is ready to transfer data, it issues a data channel request to the processor. All requests are honored according to the relative position of the requesting device on the I/O bus. The device requesting service which is physically closest to the bus \_\_\_\_\_ to the CPU is serviced first. The synchronization of new requests occurs concurrently with the honoring of other requests. The maximum transfer rate for the data channel I/O is as

Data General Corporation  
Company Confidential

Rev. 7

follows:

INPUT: 1.25 megabytes

OUTPUT: .715 megabytes

### 17.3 Burst Multiplexor I/O

The burst multiplexor channel transfers data directly between devices and the system cache via the BMC bus. The initial information for the transfer is setup via programmed I/O in the same manner as the standard data channel. The burst multiplexor channel has two address modes; physical(unmapped) and logical(mapped). When a device's request is honored by the burst multiplexor channel, the device transfers the starting address of the transfer and the number of words to be transferred to the burst multiplexor channel. As each data word is transferred to or from the cache, the address is incremented on the burst multiplexor channel. A maximum of 256 words maybe transferred for every request.

In unmapped mode, the device transmits a 21 bit address to the burst multiplexor channel. The burst multiplexor channel passes the address directly to the system cache. In mapped mode, the device passes a 20 bit logical address and specifies mapped mode when granted access to the BMC bus. The high order 10 bits of the logical address are mapped to the 14 physical address bits corres-

---

ponding to the appropriate physical page number. The physical page number is catenated with the low order 10 bits of the logical address to form the physical address. The map tables of the BMC will be described in a separate document.

A maximum of 8 devices may be attached to a burst multiplexor channel. Eagle is capable of attaching multiple burst multiplexor channels(limited by the capacity of the system cache)

The maximum transfer rate for burst multiplexor channel is as follows:

INPUT:

OUTPUT:

### 17.4 System Cache

Data General Corporation  
Company Confidential

Rev. 7



The following section is included for clarity purposes only. It does not represent an architectural specification.

The system cache of Eagle has a dedicated port for I/O devices. The overall data transfer capacity of the I/O port of the system cache is 18.2 megabytes/second. A maximum of 8 devices may be attached to this port. The width of data transfers over the cache port is 32 bits. Attached to the I/O port of the cache currently are:

- a) both the standard and the burst multiplexor channels
- b) I/O processors
- c) a multiprocessor link

The system cache is invisible to software. For a complete specification of the system cache see the memory system specification of the EAGLE IMPLEMENTATION document.

#### 17.5 I/O Processor

An I/O processor or which is program compatible (but not necessarily form compatible) is an option on Eagle. The initial implementation could use the I/O processor for the M600 if the new single board I/O processor which connects to the I/O port of the cache was not available.

#### 17.6 Multiprocessor Link

To be supplied.

Data General Corporation  
Company Confidential

Rev. 7



Data General Corporation  
Company Confidential

Rev. 7

Chapter 18  
FAIL SAFE TECHNIQUES

Methods to achieve nonstop computing and concurrent error  
diagnosing will be described in this chapter.

--End of Chapter--

Data General Corporation  
Company Confidential

## Chapter 19 ANOMALIES

Those who can not remember the past are  
condemned to repeat it.

G. Santayana.

Certain irregularities exist in the transformation of C350  
programs to the EAGLE. These anomalies are described in detail in  
this chapter.  
\_\_\_\_\_(\_\_\_\_\_)r.

### 19.1 No Load - Skip, XOP, and XOP1 Instructions

The new Eagle instructions are created from the ALC no load,  
always skip opcodes, XOP, and the XOP1 instruction. Therefore, any  
C350 program which uses these instructions will not work properly.  
This will be explained in a warning in the EAGLE Programmer's  
Reference manual.

### 19.2 Page 0 Addresses

The C350 auto increment and decrement locations (20 - 37  
octal) are disabled in EAGLE. When the Eagle ATU is enabled, they  
are used as additional hardware reserved memory locations. When  
MMPU1 or logical equals physical is enabled, they are software  
accessible.

### 19.3 PC Wraparound

The new PC is 28 bits wide and incrementing of PC due to  
straight line code is done on 28 bits. This will cause straight  
line code in location 077777 not to wrap to location 0 as it did in  
the C350.

Data General Corporation  
Company Confidential

Rev. 7



#### 19.4 Float/Fixed Conversions

Floating Point to fixed point conversions will convert the integer portion of a floating point number from the values of -32,768 (base 10) to 32,767 for 16 bit fixed point integers. Floating Point to fixed point conversions will convert the integer portion of a floating point number from the values of -2,147,483,648 (base 10) to 2,147,483,647 for double precision fixed point. Thus, the most negative integer is properly converted without MOF overflow.

— 0 —

#### 19.5 Address Wraparound

For the following Eclipse instructions, address wraparound may not occur at 077777: BAM, BLM, CMP, CMT, CMV, CTR, and EDIT. Additionally, if data movement is backwards (i.e., descending addresses) and a segment boundary is crossed a protection fault (code=4 in AC1) is signalled.

As a result of this possible anomaly, it may be possible for an Eclipse program to generate logical addresses greater than 64 KB. If this occurs, the results are undefined. This anomaly, if it exists, is on an implementation specific basis.

#### 19.6 Eclipse Signed Divide

If the result of the Eclipse signed divide is -32768 (in AC1), carry is reset to 0 indicating no overflow. On the C350 carry is set to 1. The Eagle Divide properly determines OVERFLOW (reset to 0) for the same condition.

#### 19.7 Eclipse Vector & NIO

There are three additional instruction encoding for the Eclipse Vector and NIO instructions. They are the presently defined Eclipse encoding with bits 3 and 4 being (0,1), (1,0), and (1,1). In the C350 only the (0,0) encoding is listed for the

Data General Corporation  
Company Confidential

Rev. 7

Eclipse Vector. Consequently all Eclipse INTA instructions should always use the encoding (0,0) (in bits 8 and 9) to specify unchanged for the Interrupt On Flag. Bits 8 and 9 both 1 denote the Eclipse Vector Instruction.

#### 19.8 Floating Point Fault

\_\_\_\_\_8\_\_\_\_\_ All Eagle processors respond to floating point faults upon completion of the floating point instruction that caused the fault. In the C/350, the response to a floating point fault occurs when the NEXT floating point instruction is encountered. In either case, the value of the floating point PC is the same, that is containing the address of the Floating Point instruction that caused the fault.

#### 19.9 Floating Point Numerical Algorithms

The Eclipse Floating Point loads (FLDS, FLDD) do not clean a dirty zero input. All loads simply move the memory operand to the specified FPAC. No normalization and no cleaning zero is performed. The N and Z bits of the FPSR, however, are set to reflect the loaded operand.

For all Eclipse and Eagle instructions, TRUE ZERO is guaranteed to be generated for valid inputs only. If by chance, a dirty zero is generated with invalid inputs, the result is not necessarily converted to TRUE ZERO.

The Eclipse float to fix instructions (FFAS and FFMD) leave the N and Z bits of the FPSR unchanged.

Otherwise, when bit 8 of the FPSR is a 0, the results of floating point computations performed on Eagle Processors are identical to those obtained on the C/350, for valid inputs.

#### 19.10 Eclipse Commercial Faults

An Eclipse Commercial fault loads different information in AC0, AC2, and AC3 after the fault is taken. The size of the return block, the fault code in AC1, and the meaning of the PC in the return block are identical to the results obtained on the C/350.

Rev. 7

\_\_@\_\_\_\_l Corporation

Data Genera\_

Company Confidential

## 19.11 C/350 MMPU1 Instructions

As previously noted in the instruction set and privileged instruction set chapters an attempt to execute the C/350 SYC, LMP, and MMPU1 related instructions when the Eagle ATU is enabled results in a protection violation (code=9 in AC1). Due to the nature of the Eagle ATU and logical address space, these instructions have no rational meaning when the Eagle ATU is enabled.

--End of Chapter--

Data General Corporation  
Company Confidential

Chapter 20  
C350 AND EAGLE PROGRAM COMBINATIONS

The only thing we have to fear is fear  
itself.

Franklin D. Roosevelt.

Interrupts are handled by the EAGLE Operating System, so the user may mix EAGLE and C350 instructions with a minimum of effort under certain conditions. These conditions are illustrated in this chapter.

#### 20.1 Expanding An C350 User Program To Use 32 Bits ACs

The programmer who is running under an appropriate EAGLE operating system can still choose to execute C350 programs. The programmer can also choose to expand within his process by using a select set of new EAGLE instructions to:

- \* Expand the program beyond 64KB.
- \* Use expanded data areas such\_  
\_H\_\_\_\_\_ as large arrays.
- \* Utilize the 32 bit fixed arithmetic.

To expand the program beyond 64KB, the programmer could take many alternatives. The most reliable would be to place a subroutine, which is written with new EAGLE instructions, in the high memory area. This subroutine would have to be referenced with a WJSR instruction and must follow these rules:

- \* Use EAGLE WSAVE & WRTN instructions.
- \* References to memory require new EAGLE  
memory reference type instructions.

To expand data areas by using large arrays or buffers, all address calculation must be done with 32 bit fixed integer arithmetic and all references must be through the new memory reference

Data General Corporation  
Company Confidential

Rev. 7



instructions. This can be done with spot changes to programs which will reference this data; but a more reliable method would be to create new subroutines to maintain the large arrays and reference data through these routines. Note: one potential pitfall would be to calculate a 32 bit address, then call a subroutine which does a C350 SAVE/RTN calling mechanism. This will not save the high 16 bits of the 32 bit fixed point integer accumulators.

To utilize the 32 bit fixed point arithmetic will require all operations on the data (loading, calculations, and storing) to be done with the new EAGLE instructions. This can be done with spot changes or through new subroutines, but again\_

\_\_P\_\_\_\_\_, care must be taken  
when mixing these operations with 16 bit operations.

## 20.2 EAGLE Program Calling C350 Subroutines

A programmer may wish to call an C350 subroutine from a newly created EAGLE routine. For example: new EAGLE compilers may reference old C350 runtime programs. This could only be done with a good deal of change to the C350 subroutine, and so may prove to be no better than rewriting the routine in the first place.

The required changes and reasons are as follows:

\*

Use WSAVE & WRTN.

This routine may be called from high addresses where PC is greater than 16 bits. The accumulators may contain 32 bit entities.

\*

All references outside routine may need new memory reference instruction.

Arguments passed could be 32 bit fixed point integers lower level subroutines called may be in high address space.

\*

Short negative reference on stack may require new displacements.

Use of WSAVE changes the size of the stack block pushed so that a short negative reference must be recalculated.

Data General Corporation  
Company Confidential

Rev. 7

\*

|                   |                                                                                                           |                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <u>  X  </u> Long | Routines which are<br>addresses will take<br>referenced by a JSR<br>through page zero<br>must be checked. | —<br><br>up 32 bits: it is easier<br>to run out of page zero<br>this way. WJSR must be<br>used to save the full PC. |
|-------------------|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 7

## Chapter 21 C350 Map Compatibility

When MMPU1 is enabled, the MAP conforms to the specification detailed in the document, "Eclipse C/350 Principles of Operation", 014-000610-00.

### 21.1 Eagle Instructions

When MMPU1 is enabled, an attempt to execute any Eagle instruction results in an Eclipse I/O protection violation (bit 2 of the Map status register is set to 1). The PC in the pushed narrow return block references the Eagle instruction that caused the I/O protection fault.

--End of Chapter--

Data General Corporation  
Company Confidential

Appendix I  
Instruction Set Index

22/Aug/80

Data General Corporation  
Company Confidential

Rev. 7

|             |       |       |        |      |
|-------------|-------|-------|--------|------|
| BKPT        | 7-53  |       | LDATS  | 7-12 |
|             |       |       | LDSP   | 7-41 |
| CIO         | 13-8  |       |        |      |
| CIOI        | 13-8  |       | LFAMD  | 10-4 |
| CRYTC       | 7-50  |       | LFAMS  | 10-4 |
| CRYPTO      | 7-50  |       | LFDMMD | 10-4 |
| CRYTZ       | 7-50  |       | LFDMMS | 10-4 |
| CVWN        | 7-20  |       | LFLDD  | 10-4 |
|             |       |       | LFLDS  | 10-4 |
|             |       |       | LFLST  | 10-6 |
|             |       |       | LFMMD  | 10-4 |
| DEQUE       | 9-10  |       | LFMMS  | 10-4 |
| DERR        | 7-59  |       | LFSMD  | 10-4 |
| DSZTS       | 7-12  |       | LFSMS  | 10-4 |
|             |       |       | LFSST  | 10-6 |
|             |       |       | LFSTD  | 10-4 |
|             |       |       | LFSTS  | 10-4 |
| ECLID       | 7-58  |       |        |      |
| ENQH        | 9-6   |       |        |      |
| ENQT        | 9-8   |       |        |      |
|             |       |       | LJMP   | 7-39 |
|             |       |       | LJSR   | 7-39 |
|             |       |       | LLDB   | 7-10 |
| FEXP        | 10-10 |       | LLEF   | 7-5  |
| FFAS        | 10-9  |       | LLEFB  | 7-6  |
| FFMD        | 10-9  |       | LMRF   | 12-4 |
| FLAS        | 10-10 |       | LNADD  | 7-27 |
| FRDS        | 10-8  |       | LNADI  | 7-35 |
| FRH         | 10-10 |       | LNDIV  | 7-29 |
| FTE         | 10-7  |       | LNDO   | 7-38 |
| FXTD        | 7-50  |       | LNDSZ  | 7-8  |
| FXTE        | 7-50  |       | LNISZ  | 7-8  |
|             |       |       | LNLDA  | 7-7  |
| _____h_____ | —     | LNMUL | 7-28   |      |
| ISZTS       | 7-12  |       | LNSBI  | 7-36 |
|             |       |       | LNSTA  | 7-7  |
|             |       |       | LNSUB  | 7-28 |
|             |       |       | LPEF   | 7-5  |
|             |       |       | LPEFB  | 7-6  |
| LCALL       | 8-5   |       | LPHY   | 7-61 |
| LCPID       | 7-58  |       | LPSHJ  | 7-39 |
| LCS         | 7-56  |       | LPSR   | 7-49 |
| LDAFP       | 7-11  |       |        |      |
| LDASB       | 7-11  |       |        |      |
| LDASL       | 7-11  |       |        |      |
| LDASP       | 7-11  |       | LSBRA  | 12-1 |



Data General Corporation  
Company Confidential

Rev. 7

\_\_\_\_\_p\_\_\_\_\_

Data General Corporation  
Company Confidential

Rev. 7

|         |      |       |            |
|---------|------|-------|------------|
| WBR     | 7-41 | WLDAI | 7-21       |
|         |      | WLDB  | 7-44       |
|         |      | WLDI  | 11-2       |
|         |      | WLDIX | 11-2       |
| WBSAC   | 9-11 | WLMP  | 13-3       |
| WBSAS   | 9-11 | WLOB  | 7-43       |
| WBSE    | 9-11 | WLRB  | 7-43       |
| WBSGE   | 9-11 | WLSH  | 7-23       |
| WBSLE   | 9-11 | WLSHI | 7-23       |
| WBSNE   | 9-11 | WLSI  | 7-23       |
| WBSSC   | 9-11 |       |            |
| WBSSS   | 9-11 |       |            |
| WBTO    | 7-43 |       |            |
|         |      |       |            |
| ___x___ | WBTZ | 7-43  | WLSN 11-2  |
|         |      |       | WMOV 7-16  |
|         |      |       | WMOVR 7-24 |
|         |      |       | WMSP 7-11  |
| WCIS    | 7-55 | WMUL  | 7-16       |
| WCLM    | 7-51 | WMULS | 7-16       |
| WCMP    | 7-44 | WNADI | 7-21       |
| WCMT    | 7-44 | WNEG  | 7-16       |
| WCMV    | 7-44 | WPOP  | 7-13       |
| WCOB    | 7-43 | WPOPB | 8-11       |
| WCOM    | 7-23 | WPOPJ | 7-41       |
| WCTR    | 7-44 | WPSH  | 7-13       |
| WDIV    | 7-16 | WRTN  | 8-9        |
| WDIVS   | 7-16 | WSALA | 7-47       |
| WDPOP   | 12-5 | WSALM | 7-48       |
| WEDIT   | 11-1 | WSANA | 7-47       |
|         |      | WSANM | 7-48       |
|         |      | WSAVR | 8-6        |
|         |      | WSAVS | 8-6        |
| WFFAD   | 10-9 | WSBI  | 7-21       |
| WFLAD   | 10-9 | WSCT  | 7-44       |
| WFPOP   | 10-7 | WSEQ  | 7-26       |
| WFPSH   | 10-7 | WSEQI | 7-33       |
| WFSAC   | 9-11 | WSGE  | 7-26       |
| WFSAS   | 9-11 | WSGT  | 7-26       |
| WFSE    | 9-11 | WSGTI | 7-33       |
| WFSGE   | 9-11 | WSKBO | 7-48       |
| WFSLE   | 9-11 | WSKBZ | 7-48       |
| WFSNE   | 9-11 |       |            |
| WFSSC   | 9-11 |       |            |
| WFSSS   | 9-11 |       |            |
| WHLV    | 7-16 | WSLE  | 7-26       |
| WINC    | 7-16 | WSLEI | 7-33       |
| WIOR    | 7-23 | WSLT  | 7-26       |
| WIORI   | 7-25 | WSNB  | 7-43       |

---

—

Rev. 7

Data General Corporation  
Company Confidential

WSNE 7-26  
 WSNEI 7-34  
 WSSVR 8-8  
 WSSVS 8-8  
 WSTB 7-44  
 WSTI 11-2  
 WSTIX 11-2  
 WSUB 7-16  
 WSZB 7-43  
 WSZBO 7-43  
 WUGTI 7-34  
 WULEI 7-34  
 WUSGE 7-26  
 WUSGT 7-26  
 WWCS 7-56  
 WXCH 7-16  
 WXOP 7-55  
 WXOR 7-23  
 WXORI 7-25

XCALL 8-5  
 XFAMD 10-5  
 XFAMS 10-5  
 XFDMD 10-5  
 XFDMS 10-5  
 XFLDD 10-5  
 XFLDS 10-5  
 XFMMD 10-5  
 XFMS 10-5  
 XFSMD 10-5  
 XFSMS 10-5  
 XFSTD 10-5  
 XFSTS 10-5

XJMP 7-40  
 XJSR 7-40  
 XLDB 7-10  
 XLEF 7-8  
 XLEFB 7-6  
 XNADD 7-27  
 XNADI 7-35  
 XNDIV 7-29  
 XNDO 7-38

XNISZ 7-9  
 XNLDA 7-9  
 XNMUL 7-28  
 XNSBI 7-36  
 XNSTA 7-9  
 XNSUB 7-28  
 XPEF 7-5  
 XPEFB 7-6  
 XPSHJ 7-40  
 XSTB 7-10

XVCT 12-5  
 XWADD 7-30  
 XWADI 7-35  
 XWDIV 7-32  
 XWDO 7-38  
 XWDSZ 7-9  
 XWISZ 7-9  
 XWLDA 7-8  
 XWMUL 7-31  
 XWSBI 7-36  
 XWSTA 7-8  
 XWSUB 7-30

ZEX 7-16

--End of Appendix--

\_\_\_\_\_  
 XNDSZ 7-9

Data General Corporation  
Company Confidential

Rev. 7

## APPENDIX II - INSTRUCTION ENCODINGS

The following pages contain the binary encodings of the Eclipse and Eagle instructions supported in the architecture.



Data General Corporation  
Company Confidential

EAGLE OPCODE ASSIGNMENTS  
8-07-80  
REV. 7.0

TWO ACCUMULATOR INSTRUCTIONS

| 0 | 1   | 2 | 3    | 4 | 5 | 6 | 7    | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE | LENGTH |
|---|-----|---|------|---|---|---|------|---|---|---|---|---|---|---|---|--------|--------|
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | NADD   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | NSUB   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | NMUL   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | NDIV   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WADD   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WSUB   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WMUL   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WDIV   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WADC   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WINC   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WNEG   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 0 | 1 | 1 | 1 | 1 | 0 |   |   |        |        |
|   |     |   | 0    | 1 |   |   | WASH |   |   | 1 |   |   |   |   |   |        |        |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | SEX    | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | ZEX    | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WXCH   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WMOV   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 0    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WAND   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 0    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WCOM   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WIOR   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 0    | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WXOR   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 1    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WANC   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 1    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WLSH   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 1    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WCLM   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 1    | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WPSH   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WPOP   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | WUSGE  | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | WUSGT  | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 0    | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WSEQ   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WSNE   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | WSGE   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | WSLE   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 0 | 1    | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WSGT   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WSLT   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | WBTO   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | WBTZ   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 0    | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WSZB   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WSNB   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | WSZBO  | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | WLOB   | 1      |
| 1 | ACS |   | ACD  |   | 0 | 1 | 1    | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WLRB   | 1      |
| 1 | ACS |   | ACD  |   | 1 | 0 | 0    | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WCOB   | 1      |
| 1 | ACS |   | FPAC |   | 1 | 0 | 0    | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | WFFAD  | 1      |
| 1 | ACS |   | FPAC |   | 1 | 0 | 0    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | WFLAD  | 1      |
| 1 | N   |   |      |   |   |   |      |   |   |   |   |   |   |   |   |        |        |
|   |     |   | AC   | 1 |   | 0 | 0    | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WADI   | 1      |

|   |   |    |   |   |   |   |   |   |   |   |   |   |   |      |   |
|---|---|----|---|---|---|---|---|---|---|---|---|---|---|------|---|
| 1 | N | AC | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WSBI | 1 |
| 1 | N | AC | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | NADI | 1 |
| 1 | N | AC | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | NSBI | 1 |
| 1 | N | AC | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WLSI | 1 |

## TWO ACCUMULATOR INSTRUCTIONS

\* = EFA RQD

| 0 | 1   | 2    | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE | LENGTH |
|---|-----|------|---|---|---|---|---|---|---|---|---|---|---|---|---|--------|--------|
| 1 | XR  | FPAC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | * | XFAMS  | 2      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | * | XFAMD  | 2      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XFMSMS | 2      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XFMMMD | 2      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | * | XFSMS  | 2      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | * | XFSMD  | 2      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XFDMS  | 2      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XFDMD  | 2      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | * | XFLDS  | 2      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | * | XFLDD  | 2      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XFSTS  | 2      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XFSTD  | 2      |
| 1 | XR  | AC   | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | * | XWLDA  | 2      |
| 1 | XR  | AC   | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | * | XWSTA  | 2      |
| 1 | XR  | AC   | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XNLDA  | 2      |
| 1 | XR  | AC   | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XNSTA  | 2      |
| 1 | XR  | AC   | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | * | XLEF   | 2      |
| 1 | XR  | AC   | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | * | XLDB   | 2      |
| 1 | XR  | AC   | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XSTB   | 2      |
| 1 | XR  | AC   | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XLEFB  | 2      |
| 1 | ACS | ACD  | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |   | NNEG   | 1      |
| 1 | XR  | AC   | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | * | LDSP   | 3      |
| 1 | ACS | ACD  | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |   | WLDB   | 1      |
| 1 | ACS | ACD  | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | WSTB   | 1      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LFAMS  | 3      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LFAMD  | 3      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | * | LFSMS  | 3      |
| 1 | XR  | FPAC | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | * | LFSMD  | 3      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LFMSMS | 3      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LFMMMD | 3      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LFDMS  | 3      |
| 1 | XR  | FPAC | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LFDMD  | 3      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LFLDS  | 3      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LFLDD  | 3      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LFSTS  | 3      |
| 1 | XR  | FPAC | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LFSTD  | 3      |
| 1 | XR  | AC   | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LNLDLA | 3      |
| 1 | XR  | AC   | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LNSTA  | 3      |
| 1 | XR  | AC   | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LLEF   | 3      |
| 1 | XR  | AC   | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LWLDA  | 3      |
| 1 | XR  | AC   | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LLDB   | 3      |
| 1 | XR  | AC   | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LSTB   | 3      |
| 1 | XR  | AC   | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LLEFB  | 3      |
| 1 | XR  | AC   | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LWSTA  | 3      |
| 1 | ACS | ACD  | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |   | -FREE- |        |
| 1 | ACS | ACD  | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |   | PIO    | I/O 1  |
| 1 | ACS | ACD  | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |   | CIO    | 1      |
| 1 | ACS | ACD  | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |   | CIOI   | 2      |

## TWO ACCUMULATOR INSTRUCTIONS

\* = EFA RQD

| 0        | 1    | 2   | 3   | 4 | 5 | 6    | 7   | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE  | LENGTH |
|----------|------|-----|-----|---|---|------|-----|---|---|---|---|---|---|---|---|---------|--------|
| 1        | XR   |     | AC  | 0 | 0 | 0    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNADD | 2      |
| 1        | XR   |     | AC  | 0 | 0 | 0    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNSUB | 2      |
| 1        | XR   |     | AC  | 0 | 0 | 0    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNMUL | 2      |
| 1        | XR   |     | AC  | 0 | 0 | 0    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNDIV | 2      |
| 1        | XR   |     | AC  | 0 | 0 | 1    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWADD | —      |
| ( ———— ) |      |     |     | 2 |   |      |     |   |   |   |   |   |   |   |   |         |        |
| 1        | XR   |     | AC  | 0 | 0 | 1    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWSUB | 2      |
| 1        | XR   |     | AC  | 0 | 0 | 1    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWMUL | 2      |
| 1        | XR   |     | AC  | 0 | 0 | 1    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWDIV | 2      |
| 1        | XR   |     | AC  | 0 | 1 | 0    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNADD | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 0    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNSUB | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 0    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNMUL | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 0    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNDIV | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 1    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWADD | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 1    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWSUB | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 1    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWMUL | 3      |
| 1        | XR   |     | AC  | 0 | 1 | 1    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWDIV | 3      |
| 1        | NN   |     | XR  | 1 | 0 | 0    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNADI | 2      |
| 1        | NN   |     | XR  | 1 | 0 | 0    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNSBI | 2      |
| 1        | AC   |     | XR  | 1 | 0 | 0    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XNDO  | 2      |
| 1        | FPS  |     | FPD | 1 | 0 | 0    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | FRDS    | 1      |
| 1        | NN   |     | XR  | 1 | 0 | 1    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWADI | 2      |
| 1        | NN   |     | XR  | 1 | 0 | 1    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWSBI | 2      |
| 1        | AC   |     | XR  | 1 | 0 | 1    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * XWDO  | 2      |
| 1        | XR   |     | AC  | 1 | 0 | 1    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -free-  |        |
| 1        | NN   |     | XR  | 1 | 1 | 0    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNADI | 3      |
| 1        | NN   |     | XR  | 1 | 1 | 0    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNSBI | 3      |
| 1        | AC   |     | XR  | 1 | 1 | 0    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LNDO  | 3      |
| 1        | XR   |     | AC  | 1 | 1 | 0    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -free-  |        |
| 1        | NN   |     | XR  | 1 | 1 | 1    | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWADI | 3      |
| 1        | NN   |     | XR  | 1 | 1 | 1    | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWSBI | 3      |
| 1        | AC   |     | XR  | 1 | 1 | 1    | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | * LWDO  | 3      |
| 1        | XR   |     | AC  | 1 | 1 | 1    | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -free-  |        |
| 1        | DISP | 0-3 |     | 0 |   | DISP | 4-7 |   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | WBR     |        |

# ONE ACCUMULATOR INSTRUCTIONS

\* = EFA RQD

| 0     | 1 | 2 | 3    | 4 | 5 | 6 | 7  | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE | LENGTH |   |       |   |
|-------|---|---|------|---|---|---|----|---|---|---|---|---|---|---|---|--------|--------|---|-------|---|
| 1     | 0 | 0 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | LDATS  | 1      |   |       |   |
| <hr/> |   |   |      |   |   |   |    |   |   |   |   |   |   |   |   |        |        |   |       |   |
| 0     |   |   | 1    | 0 | 0 |   | AC |   | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0      | 0      | 1 | STATS | 1 |
| 1     | 0 | 0 | -    | - | 1 | 1 | 0  | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | -FREE- |        |   |       |   |
| 1     | 0 | 0 | -    | - | 1 | 1 | 0  | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | -FREE- |        |   |       |   |
| 1     | 0 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | LDASP  | 1      |   |       |   |
| 1     | 0 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | STASP  | 1      |   |       |   |
| 1     | 0 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | LDASL  | 1      |   |       |   |
| 1     | 0 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | STASL  | 1      |   |       |   |
| 1     | 1 | 0 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | LDASB  | 1      |   |       |   |
| 1     | 1 | 0 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | STASB  | 1      |   |       |   |
| 1     | 1 | 0 | AC   |   | 1 | 1 | 0  | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | LDAFP  | 1      |   |       |   |
| 1     | 1 | 0 | AC   |   | 1 | 1 | 0  | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | STAFP  | 1      |   |       |   |
| 1     | 1 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WMSP   | 1      |   |       |   |
| 1     | 1 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WHLV   | 1      |   |       |   |
| 1     | 1 | 1 | AC   |   | 1 | 1 | 0  | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | CVWN   | 1      |   |       |   |
| 1     | 1 | 1 | FPAC |   | 1 | 1 | 0  | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WLDI   | 1      |   |       |   |

# ONE ACCUMULATOR INSTRUCTIONS

\* = EFA RQD

| 0 | 1 | 2 | 3  | 4 | 5 | 6  | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE |       |   | LENGTH |   |
|---|---|---|----|---|---|----|---|---|---|---|---|---|---|---|---|--------|-------|---|--------|---|
| 1 | 0 | 0 | XR | 1 | 1 | 0  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | * | XCALL  |       | 2 |        |   |
| 1 | 0 | 0 | XR | 1 | 1 | 0  | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | *      | XPSHJ | 2 |        |   |
| 1 | 0 | 0 | XR | 1 | 1 | 0  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XPEF   |       | 2 |        |   |
| 1 | 0 | 0 | XR | 1 | 1 | 0  | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XNISZ  |       | 2 |        |   |
| 1 | 0 | 1 | XR | 1 | 1 | 0  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | * | XNDSZ  |       | 2 |        |   |
| 1 | 0 | 1 | XR | 1 | 1 | 0  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | * | XWISZ  |       | 2 |        |   |
| 1 | 0 | 1 | XR | 1 | 1 | 0  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | * | XPEFB  |       | 2 |        |   |
| 1 | 0 | 1 | XR | 1 | 1 | 0  | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | * | XWDSZ  |       | 2 |        |   |
| 1 | 1 | 0 | XR | 1 | 1 | 0  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | * | XJMP   |       | 2 |        |   |
| 1 | 1 | 0 | XR | 1 | 1 | 0  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | * | XJSR   |       | 2 |        |   |
| 1 | 1 | 0 | AC | 1 | 1 | 0  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |   | NLDAI  |       | 2 |        |   |
| 1 | 1 | 0 | AC | 1 | 1 | 0  | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | NADDI  |       | 2 |        |   |
| 1 | 1 | 1 | AC | 1 | 1 | 0  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |   | NSALA  |       | 2 |        |   |
| 1 | 1 | 1 | AC | 1 | 1 | 0  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |   | NSALM  |       | 2 |        |   |
|   |   |   |    |   |   |    |   |   |   |   |   |   |   |   |   |        |       |   |        |   |
| 8 |   |   | 1  | 1 | 1 | AC |   | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0      | 0     | 1 | NSANA  | 2 |
| 1 | 1 | 1 | AC | 1 | 1 | 0  | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | NSANM  |       | 2 |        |   |

# ONE ACCUMULATOR INSTRUCTIONS

\* = EFA RQD

| 0 | 1 | 2 | 3    | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE | LENGTH |
|---|---|---|------|---|---|---|---|---|---|---|---|---|---|---|---|--------|--------|
| 1 | 0 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LNISZ  | 3      |
| 1 | 0 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LNDSZ  | 3      |
| 1 | 0 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LWISZ  | 3      |
| 1 | 0 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LWDSZ  | 3      |
| 1 | 0 | 1 | XR   | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LCALL  | 3      |
| 1 | 0 | 1 | XR   | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LJMP   | 3      |
| 1 | 0 | 1 | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LJSR   | 3      |
| 1 | 0 | 1 | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LPEF   | 3      |
| 1 | 1 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | * | LPSHJ  | 3      |
| 1 | 1 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | * | LFLST  | 3      |
| 1 | 1 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | * | LFSST  | 3      |
| 1 | 1 | 0 | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | * | LPEFB  | 3      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |   | WSEQI  | 2      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |   | WLSHI  | 2      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |   | WSNEI  | 2      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |   | WNADI  | 2      |
| 1 | 0 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |   | WADDI  | 3      |
| 1 | 0 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |   | WANDI  | 3      |
| 1 | 0 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |   | WIORI  | 3      |
| 1 | 0 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | WXORI  | 3      |
| 1 | 0 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |   | WSANA  | 3      |
| 1 | 0 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |   | WSALA  | 3      |
| 1 | 0 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |   | WSANM  | 3      |
| 1 | 0 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | WSALM  | 3      |
| 1 | 1 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |   | WLDAI  | 3      |
| 1 | 1 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |   | WUGTI  | 3      |
| 1 | 1 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |   | WAS_   |        |
| @ |   |   | HI   |   |   | 2 |   |   |   |   |   |   |   |   |   |        |        |
| 1 | 1 | 0 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | WULEI  | 3      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |   | WSGTI  | 2      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |   | WMOVR  | 1      |
| 1 | 1 | 1 | AC   | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |   | WSLEI  | 2      |
| 1 | 1 | 1 | FPAC | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |   | WSTI   | 1      |

# ZERO ACCUMULATOR INSTRUCTIONS

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8    | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE | LENGTH |
|---|---|---|---|---|---|---|---|------|---|---|---|---|---|---|---|--------|--------|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | PBX    | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | LCPID  | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WCTR   | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WCMV   | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WCMT   | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WCMP   | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WEDIT  | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | FXTD   | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | FXTE   | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WLDIX  | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WSTIX  | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WLSN   | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | WBLM   | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | WMULS  | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | WDIVS  | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WPOPB  | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WPOPJ  | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 1 | 1 | 0 | 0 | 1 | WRSTR  | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 0 | 1 | 0 | 0 | 1 | WRTN   | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 1 | 1 | 0 | 0 | 1 | WFPSH  | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | 1 | 0 | 0 | 1 | WFPOP  | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 1 | 1 | 0 | 0 | 1 | LPSR   | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 0 | 1 | 0 | 0 | 1 | SPSR   | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 1 | 1 | 0 | 0 | 1 | SNOVR  | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | — |   |   |   |        |        |
| H |   |   |   | 1 | 0 | 0 | 1 | BKPT |   |   |   | 1 |   |   |   |        |        |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 1 | 1 | 0 | 0 | 1 | VBP    | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 0 | 1 | 0 | 0 | 1 | VWP    | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 1 | 1 | 0 | 0 | 1 | LSBRA  | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | 1 | 0 | 0 | 1 | LSBRS  | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 1 | 1 | 0 | 0 | 1 | PATU   | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 0 | 1 | 0 | 0 | 1 | RRFB   | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 0 | 1 | 1 | 1 | 0 | 0 | 1 | ORFB   | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | LMRF   | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | SMRF   | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | LPHY   | 1      |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WDPOP  | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | CRYPTO | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | CRYTZ  | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | CRYTC  | 1      |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | WLMP   | I/O 1  |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | ISZTS  | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | DSZTS  | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | ENQH   | 1      |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | ENQT   | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 0 | 1 | 0 | 0 | 1 | DEQUE  | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 0 | 1 | 1 | 0 | 0 | 1 | LPST   | 1      |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 0 | 1 | 0 | 0 | 1 | -FREE- |        |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1    | 1 | 1 | 1 | 1 | 0 | 0 | 1 | -FREE- |        |

## ZERO ACCUMULATOR INSTRUCTION

| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0     | 1 | 2 | 3 | 4 | 5 | OPCODE    | LENGTH |
|----|---|---|---|---|---|---|---|---|---|-------|---|---|---|---|---|-----------|--------|
| 1  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 0 | 1 | 0 | 0 | 1 | WAP       | 2      |
| 1  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 1 | 1 | 0 | 0 | 1 | WCIS (1)  | 2      |
| 1  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 0 | 1 | 0 | 0 | 1 | WSSVR     | 2      |
| 1  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 1 | 1 | 0 | 0 | 1 | WSSVS     | 2      |
| 1  | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 0 | 1 | 0 | 0 | 1 | WXOP (3)  | 2      |
| 1  | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 1 | 1 | 0 | 0 | 1 | WWCS (3)  | 2      |
| 1  | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 0 | — |   |   |   |           |        |
| —P |   |   |   | 1 | 0 | 0 | 1 |   |   | WSAVR |   | 2 |   |   |   |           |        |
| 1  | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 1 | 1 | 0 | 0 | 1 | WSAVS     | 2      |
| 1  | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 0 | 1 | 0 | 0 | 1 | XVCT      | 2      |
| 1  | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 1 | 1 | 0 | 0 | 1 | SRCHQ (2) | 2      |
| 1  | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 0 | 1 | 0 | 0 | 1 | -FREE-    |        |
| 1  | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 1 | 1 | 0 | 0 | 1 | -FREE-    |        |
| 1  | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 0 | 1 | 0 | 0 | 1 | WCST      | 1      |
| 1  | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0     | 1 | 1 | 0 | 0 | 1 | -FREE-    |        |
| 1  | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 0 | 1 | 0 | 0 | 1 | -FREE-    |        |
| 1  | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1     | 1 | 1 | 0 | 0 | 1 | -FREE-    |        |
| 1  | x | x | x | 1 | 1 | 1 | 1 | 0 | 0 | x     | x | 1 | 0 | 0 | 1 | DERR      |        |

## OTHER INSTRUCTIONS

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE | LENGTH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--------|--------|
| 1 | # | # | # | 1 | 1 | 1 | 1 | 1 | 0 | # | # | 1 | 0 | 0 | 1 | WSKBZ  | 1      |
| 1 | # | # | # | 1 | 1 | 1 | 1 | 0 | 1 | # | # | 1 | 0 | 0 | 1 | WSKBO  | 1      |

NOTE 1: No instruction presently uses the OPESC encoding.

NOTE 2: The search queue encodings for the second word are:

|       |                 |   |      |   |   |
|-------|-----------------|---|------|---|---|
|       | 1               | 1 | 1    | 1 | 1 |
| 0     | 0               | 1 | 2    | 4 | 5 |
| ----- |                 |   |      |   |   |
|       | Hardware Reser. | W | Comp | D |   |

Where W=1 is wide, W=0 is narrow. Where D=0 is forwards, D=1 is backwards.  
Where comp=0 is SS, =1 is SC, =2 is AS, =3 is AC, =4 is E, =5 is GE  
=6 is LE, and =7 is NE.

NOTE 3: The WXOP and WXOP1 encodings are:

|         |     |             |   |   |  |       |   |               |          |
|---------|-----|-------------|---|---|--|-------|---|---------------|----------|
| 0       | 1   | 2           | 3 | 4 |  | 7     | 8 | 1<br>4        | 1<br>5   |
| <hr/>   |     |             |   |   |  |       |   |               |          |
| acs acd | 0   | 0           | 0 | 0 |  | XOP # | 0 | WXOP 2nd word |          |
| <hr/>   |     |             |   |   |  |       |   |               |          |
| <hr/>   |     |             |   |   |  |       |   |               |          |
|         | WCS | ENTRY POINT |   |   |  |       |   | WWCS          | 2nd word |



# NOVA ALC'S

| 0 | 1   | 2   | 3 | 4 | 5 | 6 | 7  | 8   | 9 | 0       | 1 | 2 | 3 | 4 | 5 | OPCODE |
|---|-----|-----|---|---|---|---|----|-----|---|---------|---|---|---|---|---|--------|
| 1 | ACS | ACD | 0 | 0 | 0 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | COM    |
| 1 | ACS | ACD | 0 | 0 | 1 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | NEG    |
| 1 | ACS | ACD | 0 | 1 | 0 |   | SH | CR_ |   |         |   |   |   |   |   |        |
| X |     | Y   | # |   |   |   |    |     |   |         |   |   |   |   |   |        |
| 1 | ACS | ACD | 0 | 1 | 1 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | INC    |
| 1 | ACS | ACD | 1 | 0 | 0 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | ADC    |
| 1 | ACS | ACD | 1 | 0 | 1 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | SUB    |
| 1 | ACS | ACD | 1 | 1 | 0 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | ADD    |
| 1 | ACS | ACD | 1 | 1 | 1 |   | SH | CRY | # | -SKIP-- |   |   |   |   |   | AND    |

# NOVA INSTRUCTIONS

| 0 | 1 | 2 | 3 | 4  | 5 | 6    | 7 | 8                      | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE         |
|---|---|---|---|----|---|------|---|------------------------|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 0 | 0 | 0  | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | JMP            |
| 0 | 0 | 0 | 0 | 1  | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | JSR            |
| 0 | 0 | 0 | 1 | 0  | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | ISZ            |
| 0 | 0 | 0 | 1 | 1  | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | DSZ            |
| 0 | 0 | 1 |   | AC | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | LDA            |
| 0 | 1 | 0 |   | AC | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | STA            |
| 0 | 1 | 1 |   | AC | @ | INDX |   | -----DISPLACEMENT----- |   |   |   |   |   |   |   | LEF (LEF MODE) |

# NOVA I/O INSTRUCTIONS

| 0 | 1 | 2 | 3 | 4  | 5 | 6 | 7 | 8    | 9                | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE |
|---|---|---|---|----|---|---|---|------|------------------|---|---|---|---|---|---|--------|
| 0 | 1 | 1 |   | AC | 0 | 0 | 0 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | NIO    |
| 0 | 1 | 1 |   | AC | 0 | 0 | 1 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | DIA    |
| 0 | 1 | 1 |   | AC | 0 | 1 | 0 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | DOA    |
| 0 | 1 | 1 |   | AC | 0 | 1 | 1 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | DIB    |
| 0 | 1 | 1 |   | AC | 1 | 0 | 0 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | DOB    |
| 0 | 1 | 1 |   | AC | 1 | 0 | 1 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | DIC    |
| 0 | 1 | 1 |   | AC | 1 | 1 | 0 | FUNC | --DEVICE CODE--- |   |   |   |   |   |   | DOC    |
| 0 | 1 | 1 |   | AC | 1 | 1 | 1 | TEST | --DEVICE CODE--- |   |   |   |   |   |   | SKP    |

# CPU I/O INSTRUCTIONS

| 0 | 1 | 2 | 3  | 4 | 5 | 6 | 7 | 8    | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE       |
|---|---|---|----|---|---|---|---|------|---|---|---|---|---|---|---|--------------|
| 0 | 1 | 1 | 0  | 1 | 0 | 0 | 0 | 0    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | NCLID        |
| 0 | 1 | 1 | 1  | 0 | 0 | 0 | 0 | 0    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | LCS          |
| 0 | 1 | 1 | AC |   | 0 | 0 | 0 | 0    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | INTEN        |
| 0 | 1 | 1 | AC |   | 0 | 0 | 0 | 1    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | INTDS        |
| 0 | 1 | 1 | AC |   | 0 | 1 | 1 | 1    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | VCT          |
| 0 | 1 | 1 | AC |   | 0 | 0 | 1 | FUNC |   | 1 | 1 | 1 | 1 | 1 | 1 | READS        |
| 0 | 1 | 1 | AC |   | 0 | 1 | 0 | FUNC |   | 1 | 1 | 1 | 1 | 1 | 1 | HALTA        |
| 0 | 1 | 1 | AC |   | 0 | 1 | 1 | FUNC |   | 1 | 1 | 1 | 1 | 1 | 1 | INTA Note 1) |
| 0 | 1 | 1 | AC |   | 1 | 0 | 0 | FUNC |   | 1 | 1 | 1 | 1 | 1 | 1 | MSKO         |
| 0 | 1 | 1 | AC |   | 1 | 0 | 1 | FUNC |   | 1 | 1 | 1 | 1 | 1 | 1 | IORST        |
| 0 | 1 | 1 | AC |   | 1 | 1 | 1 | TEST |   | 1 | 1 | — |   |   |   |              |
| — | — | — | —  | 1 | 1 | 1 | 1 |      |   |   |   |   |   |   |   | CPUSKP       |

# MMPU I/O INSTRUCTIONS

| 0 | 1 | 2 | 3  | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE     |
|---|---|---|----|---|---|---|---|---|---|---|---|---|---|---|---|------------|
| 0 | 1 | 1 | AC |   | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | NIOP MAP   |
| 0 | 1 | 1 | AC |   | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | DIA AC MAP |
| 0 | 1 | 1 | AC |   | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | DOA AC MAP |
| 0 | 1 | 1 | AC |   | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | DOB AC MAP |
| 0 | 1 | 1 | AC |   | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | DIC AC MAP |
| 0 | 1 | 1 | AC |   | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | DOC AC MAP |

# BMC I/O INSTRUCTIONS

| 0 | 1 | 2 | 3  | 4 | 5 | 6 | 7 | 8    | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE       |
|---|---|---|----|---|---|---|---|------|---|---|---|---|---|---|---|--------------|
| 0 | 1 | 1 | AC |   | 0 | 0 | 0 | FUNC |   | 0 | 0 | 0 | 1 | 0 | 1 | NOP F AC BMC |
| 0 | 1 | 1 | AC |   | 0 | 1 | 0 | FUNC |   | 0 | 0 | 0 | 1 | 0 | 1 | DOA F AC BMC |
| 0 | 1 | 1 | AC |   | 1 | 0 | 0 | FUNC |   | 0 | 0 | 0 | 1 | 0 | 1 | DOB F AC BMC |
| 0 | 1 | 1 | AC |   | 1 | 0 | 1 | FUNC |   | 0 | 0 | 0 | 1 | 0 | 1 | DIC F AC BMC |
| 0 | 1 | 1 | AC |   | 1 | 1 | 0 | FUNC |   | 0 | 0 | 0 | 1 | 0 | 1 | DOC F AC BMC |

Note 1)Please see the description of the INTA instruction for proper use of the FUNC field.

# ECLIPSE

| 0 | 1   | 2 | 3   | 4   | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE       |
|---|-----|---|-----|-----|---|---|---|---|---|---|---|---|---|---|---|--------------|
| 1 | N   |   | ACS | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | ADI          |
| 1 | N   |   | ACS | 0   | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | SBI          |
| 1 | ACS |   | ACD | 0   | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | DAD          |
| 1 | ACS |   | ACD | 0   | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | DSB          |
| 1 | ACS |   | ACD | 0   | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | IOR          |
| 1 | ACS |   | ACD | 0   | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | XOR          |
| 1 | ACS |   | ACD | 0   | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | ANC          |
| 1 | ACS |   | ACD | 0   | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | XCH          |
| 1 | ACS |   | ACD | 0   | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | SGT          |
| 1 | ACS |   | ACD | 0   | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | SGE          |
| 1 | ACS |   | ACD | 0   | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | LSH          |
| 1 | ACS |   | ACD | 0   | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | DLSH         |
| 1 | N   |   | ACS | 0   | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | HXL          |
| 1 | N   |   | ACS | 0   | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | HXR          |
| 1 | N   |   | ACS | 0   | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | DHXL         |
| 1 | N   |   | ACS | 0   | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | DHXR         |
| 1 | ACS |   | ACD | 1   | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | BTO          |
| 1 | -   |   |     |     |   |   |   |   |   |   |   |   |   |   |   |              |
| h |     |   | ACS | ACD | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | BTZ          |
| 1 | ACS |   | ACD | 1   | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | SZB          |
| 1 | ACS |   | ACD | 1   | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | SZBO         |
| 1 | ACS |   | ACD | 1   | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | LOB          |
| 1 | ACS |   | ACD | 1   | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | LRB          |
| 1 | ACS |   | ACD | 1   | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | COB          |
| 1 | ACS |   | ACD | 1   | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | LDB          |
| 1 | ACS |   | ACD | 1   | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | STB          |
| 1 | ACS |   | ACD | 1   | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | PSH          |
| 1 | ACS |   | ACD | 1   | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |   | POP          |
| 1 | ACS |   | ACD | 1   | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | XOP0 2 words |
| 1 | 0   | 0 | 0   | 0   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |              |
| 1 | 0   | 0 | 0   | 1   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | DPOP (NOP)   |
| 1 | 0   | 0 | 1   | 0   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | LMP ****     |
| 1 | 0   | 0 | 1   | 1   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |              |
| 1 | 0   | 1 | -   | -   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |              |
| 1 | 1   | 0 | -   | -   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |              |
| 1 | 1   | 1 | -   | -   | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |              |
| 1 | ACS |   | ACD | 1   | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |   | SYC          |
| 1 | -   | - | -   | -   | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |              |
| 1 | 0   | 0 | 0   | 0   | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | PSHR         |
| 1 | 0   | 0 | 0   | 1   | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | POPB         |
| 1 | 0   | 0 | 1   | 0   | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | BAM          |
| 1 | 0   | 0 | 1   | 1   | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | POPJ         |
| 1 | 0   | 1 | 0   | 0   | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | SAVZ         |

The second word of Eclipse XOP0 is:

|              |       |   |
|--------------|-------|---|
| 1            | 1     | 1 |
| 0            | 0     | 1 |
| -----        |       |   |
| 000000000000 | XOP # |   |
| -----        |       |   |

|   |   |     |   |     |   |   |           |   |   |   |   |      |   |   |   |                   |
|---|---|-----|---|-----|---|---|-----------|---|---|---|---|------|---|---|---|-------------------|
| 1 | 0 | 1   | 0 | 1   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | RTN               |
| 1 | 0 | 1   | 1 | 0   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | BLM               |
| 1 | 0 | 1   | 1 | 1   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | DIVX              |
| 1 | 1 | 0   | 0 | 0   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | MUL               |
| 1 | 1 | 0   | 0 | 1   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | MULS              |
| 1 | 1 | 0   | 1 | 0   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | DIV               |
| 1 | 1 | 0   | 1 | 1   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | DIVS              |
| 1 | 1 | 1   | 0 | 0   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | SAVE I            |
| 1 | 1 | 1   | 0 | 1   | 1 | 1 | 1         | 1 | — |   |   |      |   |   |   |                   |
| p |   |     |   | 1   | 0 | 0 | 1         | 0 | 0 | 0 |   | RSTR |   |   |   |                   |
| 1 | 1 | 1   | 1 | 0   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 |                   |
| 1 | 1 | 1   | 1 | 1   | 1 | 1 | 1         | 1 | 1 | 0 | 0 | 1    | 0 | 0 | 0 | ECLID (see LCPID) |
| 1 |   | ACS |   | ACD |   |   | OPERATION |   |   | 0 | 1 | 1    | 0 | 0 | 0 | Eagle Ops         |
| 1 |   | ACS |   | ACD | 0 |   | OPERATION |   |   | 1 | 1 | 1    | 0 | 0 | 0 | Eagle Ops (WBR)   |
| 1 | 0 | 0   | 0 | 0   | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | EJMP              |
| 1 | 0 | 0   | 0 | 1   | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | EJSR              |
| 1 | 0 | 0   | 1 | 0   | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | EISZ              |
| 1 | 0 | 0   | 1 | 1   | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | EDSZ              |
| 1 | 0 | 1   |   | AC  | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | ELDA              |
| 1 | 1 | 0   |   | AC  | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | ESTA              |
| 1 | 1 | 1   |   | AC  | 1 |   | XR        | 0 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | ELEF              |
| 1 | 0 | 0   |   | AC  | 1 |   | XR        | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | ELDB              |
| 1 | 0 | 1   |   | AC  | 1 |   | XR        | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | ESTB              |
| 1 | 1 | 0   |   | AC  | 1 |   | XR        | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | DSPA              |
| 1 | 1 | 1   |   | --  | 1 |   | --        | 0 | 1 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 | 0 | 0   | 0 | 0   | 1 |   | XR        | 1 | 0 | 1 | 1 | 1    | 0 | 0 | 0 | PSHJ              |
| 1 | 0 | 0   | 0 | 1   | 1 | - | -         | 1 | 0 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 | 0 | 0   | 1 | 0   | 1 | - | -         | 1 | 0 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 | 0 | 0   | 1 | 1   | 1 | - | -         | 1 | 0 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 | 0 | 1   | - | -   | 1 | - | -         | 1 | 0 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 | 1 | -   | - | -   | 1 | - | -         | 1 | 0 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 |   | ACS |   | ACD | 1 | 0 | 0         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | CLM               |
| 1 |   | ACS |   | ACD | 1 | 0 | 1         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | SNB               |
| 1 | 0 | 0   |   | AC  | 1 | 1 | 0         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | MSP               |
| 1 | 0 | 1   |   | AC  | 1 | 1 | 0         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | XCT               |
| 1 | 1 | 0   |   | AC  | 1 | 1 | 0         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | HLV               |
| 1 | 1 | 1   | - | -   | 1 | 1 | 0         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 |                   |
| 1 | 0 | 0   |   | AC  | 1 | 1 | 1         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | IORI              |
| 1 | 0 | 1   |   | AC  | 1 | 1 | 1         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | XORI              |
| 1 | 1 | 0   |   | AC  | 1 | 1 | 1         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | ANDI              |
| 1 | 1 | 1   |   | AC  | 1 | 1 | 1         | 1 | 1 | 1 | 1 | 1    | 0 | 0 | 0 | ADDI              |

# ECLIPSE FLOATING POINT

| 0 | 1    | 2 | 3    | 4    | 5    | 6    | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | OPCODE    |
|---|------|---|------|------|------|------|---|---|---|---|---|---|---|---|---|-----------|
| 1 | FACS |   | FACD | 0    | 0    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FAS       |
| 1 | FACS |   | FACD | 0    | 0    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FAD       |
| 1 | FACS |   | FACD | 0    | 0    | 0    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSS       |
| 1 | FACS |   | FACD | 0    | 0    | 0    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSD       |
| — | x    |   | 1    | FACS | FACD | 0    | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 0 0 FMS |
| 1 | FACS |   | FACD | 0    | 0    | 1    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FMD       |
| 1 | FACS |   | FACD | 0    | 0    | 1    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FDS       |
| 1 | FACS |   | FACD | 0    | 0    | 1    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FDD       |
| 1 | XR   |   | FPAC | 0    | 1    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FAMS      |
| 1 | XR   |   | FPAC | 0    | 1    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FAMD      |
| 1 | XR   |   | FPAC | 0    | 1    | 0    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSMS      |
| 1 | XR   |   | FPAC | 0    | 1    | 0    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSMD      |
| 1 | XR   |   | FPAC | 0    | 1    | 1    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FMMS      |
| 1 | XR   |   | FPAC | 0    | 1    | 1    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FMMD      |
| 1 | XR   |   | FPAC | 0    | 1    | 1    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FDMS      |
| 1 | XR   |   | FPAC | 0    | 1    | 1    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FDMD      |
| 1 | XR   |   | FPAC | 1    | 0    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FLDS      |
| 1 | XR   |   | FPAC | 1    | 0    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FLDD      |
| 1 | XR   |   | FPAC | 1    | 0    | 0    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSTS      |
| 1 | XR   |   | FPAC | 1    | 0    | 0    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSTD      |
| 1 | AC   |   | FPAC | 1    | 0    | 1    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FLAS      |
| 1 | XR   |   | FPAC | 1    | 0    | 1    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FLMD      |
| 1 | AC   |   | FPAC | 1    | 0    | 1    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FFAS      |
| 1 | XR   |   | FPAC | 1    | 0    | 1    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FFMD      |
| 1 | 0    | 0 | FPAC | 1    | 1    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FNOM      |
| 1 | 0    | 1 | FPAC | 1    | 1    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FRH       |
| 1 | 1    | 0 | FPAC | 1    | 1    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FAB       |
| 1 | 1    | 1 | FPAC | 1    | 1    | 0    | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |   | FNEG      |
| 1 | 0    | 0 | FPAC | 1    | 1    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSCAL     |
| 1 | 0    | 1 | FPAC | 1    | 1    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FEXP      |
| 1 | 1    | 0 | FPAC | 1    | 1    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FINT      |
| 1 | 1    | 1 | FPAC | 1    | 1    | 0    | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FHLV      |
| 1 | 0    | 0 | 0    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FNS       |
| 1 | 0    | 0 | 0    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSA       |
| 1 | 0    | 0 | 1    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSEQ      |
| 1 | 0    | 0 | 1    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNE      |
| 1 | 0    | 1 | 0    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSLT      |
| 1 | 0    | 1 | 0    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSGE      |
| 1 | 0    | 1 | 1    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSLE      |
| 1 | 0    | 1 | 1    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSGT      |
| 1 | 1    | 0 | 0    | 0    | 0    | 1    | 1 | 0 | 1 | 0 | 1 | — |   |   |   |           |
|   |      |   | 0    | 0    | 0    | FSNM |   |   |   |   |   |   |   |   |   |           |
| 1 | 1    | 0 | 0    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSND      |
| 1 | 1    | 0 | 1    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNU      |
| 1 | 1    | 0 | 1    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNUD     |
| 1 | 1    | 1 | 0    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNO      |
| 1 | 1    | 1 | 0    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNOD     |
| 1 | 1    | 1 | 1    | 0    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNUO     |
| 1 | 1    | 1 | 1    | 1    | 1    | 1    | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |   | FSNER     |
| 1 | 0    | 0 | XR   | 1    | 1    | 0    | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   | FSST      |

|   |      |      |      |   |   |   |   |   |   |   |   |   |   |   |       |    |
|---|------|------|------|---|---|---|---|---|---|---|---|---|---|---|-------|----|
| 1 | 0    | 1    | XR   | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | FLST  |    |
| 1 | 1    | 0    | 0    | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FTE   |    |
| 1 | 1    | 0    | 0    | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FTD   |    |
| 1 | 1    | 0    | 1    | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FCLE  |    |
| 1 | 1    | 0    | 1    | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |
| 1 | 1    | 1    | 0    | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FPSH  |    |
| 1 | 1    | 1    | 0    | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FPOP  |    |
| 1 | 1    | 1    | 1    | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |
| 1 | 1    | 1    | 1    | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |
| 1 | FACS | FACD | 1    | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | FCMP  |    |
| 1 | FACS | FACD | 1    | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | FMOV  |    |
| 1 | 0    | 0    | FPAC | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | LDI   |    |
| 1 | 0    | 1    | FPAC | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | STI   |    |
| 1 | 1    | 0    | 0    | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | LDIX  |    |
| 1 | 1    | 0    | 0    | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | STIX  |    |
| 1 | 1    | 0    | 1    | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | CMV   |    |
| 1 | 1    | 0    | 1    | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | CMP   |    |
| 1 | 1    | 1    | 0    | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | CTR   |    |
| 1 | 1    | 1    | 0    | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | CMT   |    |
| 1 | 1    | 1    | 1    | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | EDIT  |    |
| 1 | 1    | 1    | 1    | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | LSN   |    |
| 1 | 0    | 0    | 0    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FLOGS | NS |
| 1 | 0    | 0    | 0    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FSQRS | NS |
| 1 | 0    | 0    | 1    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |
| 1 | 0    | 0    | 1    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FLOGD | NS |
| 1 | 0    | 1    | 0    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FPLYD | NS |
| 1 | 0    | 1    | 0    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FEXPS | NS |
| 1 | 0    | 1    | 1    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FSIND | NS |
| 1 | 0    | 1    | 1    | 1 | 1 | 1 | 1 | 1 |   |   |   |   |   |   |       |    |
|   |      |      |      | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |   |   |   | FSINS | NS |
| 1 | 1    | 0    | 0    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FPLYS | NS |
| 1 | 1    | 0    | 0    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FEXPD | NS |
| 1 | 1    | 0    | 1    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FCOSD | NS |
| 1 | 1    | 0    | 1    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FCOSS | NS |
| 1 | 1    | 1    | 0    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | FSQRD | NS |
| 1 | 1    | 1    | 0    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |
| 1 | 1    | 1    | 1    | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |
| 1 | 1    | 1    | 1    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |       |    |

NS - Not Supported

### Appendix III Scientific Instruction Set

The following is chapter 1 of the SIS instruction set. The complete document is very extensive and lengthy. A copy may be obtained from Arthur John.

Data General Corporation  
Company Confidential

14:18:43  
2/Sep/80  
Rev. 7

III-2

14:18:43  
2/Sep/80  
Rev. 7



Data General Corporation  
Company Confidential

Chapter 1  
The Eagle Scientific Instruction Set

This document specifies the Scientific Instruction Set (SIS). SIS is comprised of two components : 1) VIS - Vector Instruction Set and 2) IIS - Intrinsic Instruction Set. Chapter 2 will provide an extensive specification of VIS and chapter 3 will\_

---

do the same  
for IIS.

SIS is intended to accelerate high-level language performance for Eagle scientific applications by extending the Eclipse Floating Point Functions and Integral Array Processing to the Eagle architecture.

### 1.1 Overview of the Vector Instruction Set

The Vector Instruction Set provides high performance processing capability for array processing applications which do not require or desire signal processing instructions. Therefore VIS will be similar to the Eclipse AP/130, and S/250-IAP systems with the following exceptions: 1) VIS will not provide signal processing instructions (like FFT, Convolution, Recursive Filter, etc.) and 2) VIS will provide 16-bit and 32-bit integer, double precision floating point, single and double precision complex support for the Eagle class machines.

The objectives of VIS are:

- \* VIS will be interruptable and use machine state that is currently architecturally defined, therefore allowing immediate XYZZY (MAGIC) support.
- \* VIS will provide Fortran 77 with the capability of supporting the proposed Array Processing extensions when these standards are adopted, and presently, PL/1.
- \* VIS will be support by run-time software (APS) for Fortran, and DG/L, via CLRE.

### 1.2 Overview of the Intrinsic Instruction Set

The Intrinsic Instruction Set provides instructions which enable high-level languages, such as Fortran, DG/L, and PL/I, to significantly increase performance in scalar computation intensive

8:57:42  
6/Aug/80  
Rev. 3\_

\_\_\_\_\_.00

Data General Corporation  
Company Confidential

programs. These instructions are compatible with the Eclipse Floating Point Function option offered on various Eclipse models. Single and double precision floating point format support is provided.

The objectives of IIS are:

- \* IIS accelerates, by replacing, the traditional FORTRAN intrinsic routines.
- \* IIS instructions are "in-line", i.e. the instruction should not require subroutine linkages to operate correctly.
- \* IIS does not require its coefficients to be part of the instruction stream.
- \* IIS allows accumulator input/output usage flexibility.

### 1.3 Overview of Illegal and Unimplemented Instructions

SIS compatibility must be maintained between all Eagle models. This compatibility will be provided via the UIT (Unimplemented Instruction Trap). When a SIS instruction is encountered which is not supported by firmware a wide return block will be pushed onto the stack and the instruction will be trapped by an indirect jump to the indirect address contained in 3 + page 0 location 15. AC0 will be loaded with the second word of the SIS opcode which caused the trap. AC1, AC2, and AC3 will remain unchanged. Control will be passed to a software emulator which will perform the requested operation.

The emulator will be designed such that it exactly follows all SIS conventions defined within this specification. Additionally, the emulator must return the identical numeric results as the firmware implementation returns. Thus, consistent arithmetic

— | operations will be maintained by all Data General processors which

| support SIS whether it be through firmware, hardware or the emulator.

| If UIT has been entered because of an undefined instruction, a no-op will be performed, and control will return to the location following the undefined op-code.

6/Aug/80  
Rev. 3.00

Data General Corporation  
Company Confidential

#### 1.4 Reserved Registers and Memory Locations

SIS uses nine Eagle hardware registers and six page zero locations (Addresses are in Octal radix):

- \* AC0 is used by SIS to return the address of an instruction which caused an error.
- \* AC1 is used to return SIS error codes.
- \* AC2 is used by VIS as a pointer to the base of the Parameter Block, and return the element index of where a floating point error occurred.
- \* PSR is updated by fixed point VIS instructions
- \* FPAC0 - FPAC3 are the floating point accumulators used by IIS
- \* FPSR - is the 64 bit floating point status register updated by floating point VIS and IIS instructions
- \* Locations 15: UIT Handler Pointer. Indirect.
- \* Locations 32-33: IIS Coefficient Table Pointer. Indirectable.
- \* Location 37: Fixed Point Fault Handler Pointer. Indirectable.
- \* Location 45: Floating Point Fault Handler Pointer. Indirectable.

All locations \_  
 \_ (\_\_\_\_\_ are octal. All pointers are always interpreted as within the current segment. Thus, bits 1-3 of the double word fetched are ignored. Locations 30-33 are valid for every segment other than 0 (Ring 0). In Ring 0 these locations have other meanings. (This assumes that the operating system code which resides in Ring 0 never executes a SIS instruction.)

The coefficient table is constant for all models, although its usage is model dependent. Therefore the processor may or may not use the table data. It is anticipated that some models will provide an internal capability for providing this table. For these models locations 32,33 will be ignored. All other models must provide a table organized as follows:

Data General Corporation  
Company Confidential

Rev. 3.00

Table 1-1. IIS Coefficient Table

|   |                                                    |            |
|---|----------------------------------------------------|------------|
|   | * Single precision SINE/COSINE coefficients        | Increasing |
|   | * Double precision SINE/COSINE coefficients        | memory     |
|   | * Single precision ARCSINE/ARCCOSINE coefficients  |            |
|   | * Double precision ARCSINE/ARCCOSINE coefficients  |            |
|   | * Single precision HYPSINE/HYPCOSINE coefficients  |            |
|   | * Double precision HYPSINE/HYPCOSINE coefficients  |            |
|   | * Single precision TANGENT coefficients            |            |
|   | * Double precision TANGENT coefficients            |            |
|   | * Single precision ARCTANGENT coefficients         |            |
|   | * Double precision ARCTANGENT coefficients         |            |
|   | * Single pr_                                       |            |
| 0 | recision ARCTANGENT2 coefficients                  |            |
|   | * Double precision ARCTANGENT2 coefficients        |            |
|   | * Single precision HYPTANGENT coefficients         |            |
|   | * Double precision HYPTANGENT coefficients         |            |
|   | * Single precision Natural/Common LOG coefficients |            |
|   | * Double precision Natural/Common LOG coefficients |            |
|   | * Single precision EXPONENTIAL coefficients        |            |
|   | * Double precision EXPONENTIAL coefficients        |            |
|   | * Single precision SQUARE ROOT coefficients        |            |
|   | * Double precision SQUARE ROOT coefficients        |            |

The actual values used for the coefficient table may be found in appendix A of this document.

## 1.5 SIS Algorithms

### 1.5.1 Accuracy and Arithmetic Algorithms

SIS algorithms have the following characteristics.

All floating point inputs are assumed to have the following structure unless otherwise noted by an instruction:

- 1) All inputs are normalized.
- 2) All inputs with a zero mantissa are assumed to also have a zero sign bit and an all zero exponent (True Zero).

Data General Corporation  
Company Confidential

Rev. 3.00



All single precision operations which specify floating point accumulators (FPAC0 - FPAC3) fetch the most significant 32 bits of the FPAC and ignore the least significant 32 bits. Upon completion of the operation, the result is returned to the most significant port\_

8\_\_\_\_\_ion of the FPAC, and the least significant 32 bits of the FPAC are set to all 0's.

All floating point intermediate results within an atomic arithmetic operation (i.e. add, subtract, multiply, divide) always maintain a mantissa of 28 or 32 bits for single precision and 60 or 64 bits for double precision, depending upon the state of FPSR<8>. The intermediate result is then rounded or truncated after completion and stored in the destination, either memory or FPAC, or an internal temporary register, as either 32 (single precision) or 64 (double precision) bits. The accuracy of the results produced by firmware is identical to that produced by software emulation.

## 1.5.2 Truncation and Rounding Options

All SIS floating point operations use the Eagle rounding algorithm, i.e. FPSR<8> determines whether the result will be rounded or truncated. When FPSR<8> is a 1, the rounding algorithm is an "Unbiased Round". The two guard digits are examined as one 8 bit quantity. If their binary value is less than 128, the result is truncated. If their binary value is greater than 128, a 1 is added to the least significant bit of the truncated result. If their value is equal to 128, the least significant bit of the truncated result is added to the truncated result.

When FPSR<8> is a 0, one guard digit is used for intermediate calculations. The result is then truncated after normalization to a 24 or 56 bit mantissa for single or double precision operations, and then stored into the destination.

## 1.6 SIS Error Handling Overview

### 1.6.1 Floating Point Errors

SIS extends the Eagle floating point fault mechanisms. Both VIS and IIS detect and respond to floating point errors much in the same fashion as do the Floating Point Ins\_

@\_\_\_\_\_tructions, however SIS

6/Aug/80  
Rev. 3.00

Data General Corporation  
Company Confidential

adds new dimensions. Due to the nature of VIS operating on arrays, it is necessary to provide control for detection and to optionally ignore exponent overflows, underflows, and to detect other errors. FPSR has been expanded to provide this capability. If, during a VIS instruction, an exponent error is detected, then FPSR will be examined. If "continue on error" (bits 10 or 11) is set, FPSR will NOT be updated and processing will continue. Otherwise if FPSR<5> is set the instruction will terminate, loading AC2 with the index of the element which caused the error, and the standard Floating Point Fault mechanism will be invoked. See Chapter 2 for a detailed discussion of the enhancements to FPSR.

#### 1.6.1.1 SIS Traps

VIS and IIS perform checks on input values prior to certain operations. If these values do not meet the requirements, a wide return block is pushed onto the program stack, control transfers to the location indicated by page zero location 45, AC0 is loaded with the address of the SIS instruction that caused the error and AC1 is loaded with an error code. AC2 and AC3 remain unchanged. This mechanism is the SIS trap.

Note -- The original contents of accumulators are only altered by SIS if an error condition occurs.

#### 1.6.2 Integer Errors

SIS conforms to the Eagle fixed point fault mechanisms. Unlike VIS floating point err\_

\_\_\_\_H\_\_\_\_ors, when operating on fixed point

arrays if a fixed point error occurs erroneous answers will be stored in the destination and the instruction will continue. At the completion of the operation PSR<1> will be updated, and the instruction will terminate, trapping if PSR<0> is set. See section 2.3.2 for further discussion on fixed point errors.

--End of Chapter--

Data General Corporation  
Company Confidential

Rev. 3.00

\_\$